Elevate Web Builder 3 Manual

Table Of Contents

Chapter 1
Getting Started

1.1 System Requirements

IDE Requirements

The Elevate Web Builder IDE requires the following:

- Windows 7 or higher (Windows 10 recommended)
- 1024x768 or higher display resolution (widescreen display highly recommended)
- 32-bit color display adapter
- 1GB of installed memory
- 512MB of disk space
- Internet Explorer 11 or higher

Runtime Browser Compatibility

Applications created with Elevate Web Builder will work with Internet Explorer 11 or higher, as well as any version of the Edge, Firefox, Chrome, Safari, or Opera browsers available since 2015.

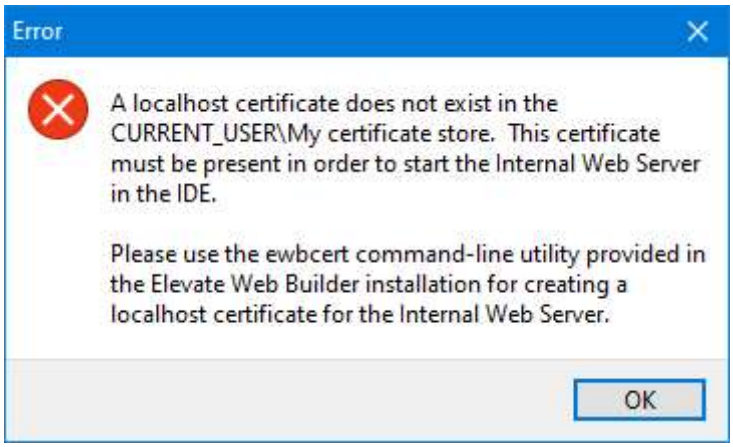
1.2 Before Using the Product

Internal Web Server Certificate Installation

The IDE requires a localhost certificate to be installed for the current user in order to support secure HTTPS browser connections when running client browser applications in the IDE or from an external browser.

> **Warning**
> This localhost certificate should only be used for development purposes because it is not signed by a valid certificate authority.

The internal web server embedded in the IDE will not start properly without this localhost certificate being installed, and you will see an error during the IDE startup if it is not present:



The **ewbcert** utility is included to automate the process of creating the proper localhost certificate for use with the IDE. You will find it in the following installation location:

```
C:\Program Files (x86)\Elevate Web Builder 3\bin\ewbcert\win32\ewbcert.exe
```

In order to allow the localhost certificate to work correctly with all modern browsers, the ewbcert utility first creates a root certificate called "EWBRootCA", and then uses that root certificate to create/sign the localhost certificate.

Command-line usage:

```
ewbcert <Certificate Name> [-s<Certificate Store Name>] [-t<Certificate Store
      Type>] [-c|-r]

Descriptions:
-s Certificate store name, enclose in double quotes (default is "My")
-t Certificate store type: CURRENT_USER or LOCAL_MACHINE (default is
      CURRENT_USER)
-c|-r Create/remove the certificate (default is to check for certificate)
Exit codes are 0 for success, 1 for failure
```

The following command can be used to create the localhost certificate for the IDE:

```
ewbcert localhost -c
```

The following command can be used to remove the localhost certificate for the IDE:

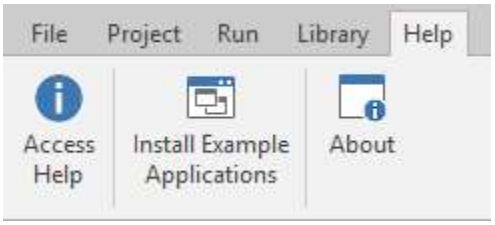
```
ewbcert localhost -r
```

Note
If you attempt to create a localhost certificate in a LOCAL_MACHINE certificate store, you will need to do so using a command prompt running with administrator privileges or the certificate creation will fail with an "Access denied" error.

Installing the Example Applications

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

- Click on the **Help** tab on the main menu. The Help menu will appear:



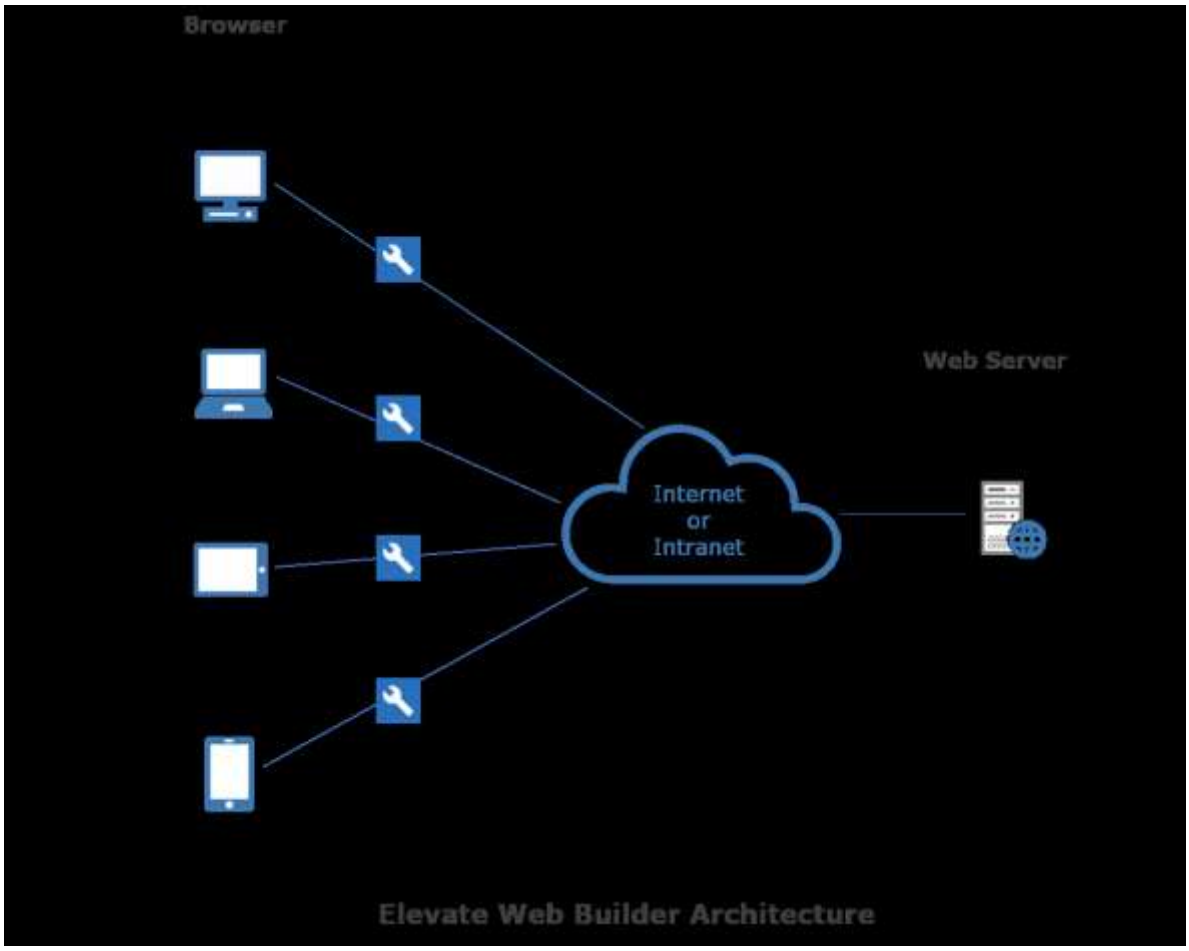
- Click on the **Install Example Applications** button on the Help menu. This will start the process of copying the example applications to the following directory for the current user account:

```
My Documents\Elevate Web Builder 3\Projects
```

1.3 General Architecture

Elevate Web Builder allows developers to easily create compact and dynamic single-page web browser applications that rival the functionality of desktop applications, as well as web server applications that can provide API access to content, data, and other back-end functionality, all using the same source language and syntax.

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



The web server application can be any back-end application (PHP, .NET, Python, Go, etc.) that can be called using a server request to a web server (IIS, Apache, NGINX, etc.) and return a response. However, the Elevate Web Builder IDE has been specifically designed to work seamlessly with the included Elevate Web Builder Web Server and Elevate Web Builder server applications that are created, compiled, and deployed within the IDE. The Elevate Web Builder Web Server includes a runtime that allows you to execute and remotely debug the Elevate Web Builder server applications, allowing you to create and execute both client and server applications using the same source language in the same IDE. In addition, the Elevate Web Builder Web Server includes the ability to deploy and execute native server modules created using Delphi XE2 or higher. This is a good option for developers that have a large investment in an existing Delphi codebase and want to re-use as much of the existing code as possible without making any significant changes.

The Elevate Web Builder Web Server includes the following functionality:

- Complete TLS (Transport Level Security) support for secure access, including the ability to force the redirection of all insecure requests to corresponding secure requests.

- Complete session support, including the ability to control the length of the cryptographically-randomized session IDs (the session IDs are sent to the user agent using HTTP-only cookies in order to prevent cross-site scripting attacks).
- Role-based access control, including the ability to create custom privileges, assign privileges to any server applications/native server modules or dataset commands, activate/deactivate roles and users, lock/unlock users for a period of time manually or based upon N authentication failures, and restrict the time of day when a user can log in.
- Both HTTP request logging (CSV) and application-level audit logging (JSON), including the ability to download logs directly from the IDE for further analysis.
- Server monitoring for various statistics (reads/writes per second, number of sessions, etc.) directly from the IDE.
- A REST API for database access that automatically prevents SQL injection attacks and offers nested transactions and support for complex, database engine-specific SQL statements.
- Complete content management, including the ability to upload/deploy client or server applications/native server modules with a single click, upload/download arbitrary content or files, and create and delete folders.
- Server migration functionality that allows you to migrate all or a subset of server objects from one web server to another.
- Remote debugging of server applications, including breakpoints and step/step into/step over functionality.

Note
Most of the above-mentioned functionality is accessible using an administration API that the Elevate Web Builder IDE uses to perform the remote administration of the Elevate Web Builder Web Server. While the API is specific to the Elevate Web Builder Web Server, it is still a standard HTTP API and could also be implemented using a different web server or web server application platform such as Apache and PHP. The Elevate Web Builder IDE only requires that the implemented web server API conform to the required specifications, and has no specific implementation bindings to the Elevate Web Builder Web Server.

Please see the following topics for more information on the Elevate Web Builder Web Server:

Configuring the Web Server
Starting the Web Server
Web Server Request Handling
Web Server Security
Web Server Logging
Web Server Authentication
Web Server Database Access
Web Server Applications
Web Server Native Modules
Web Server Administration

Core Language

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

Integrated Development Environment (IDE)

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

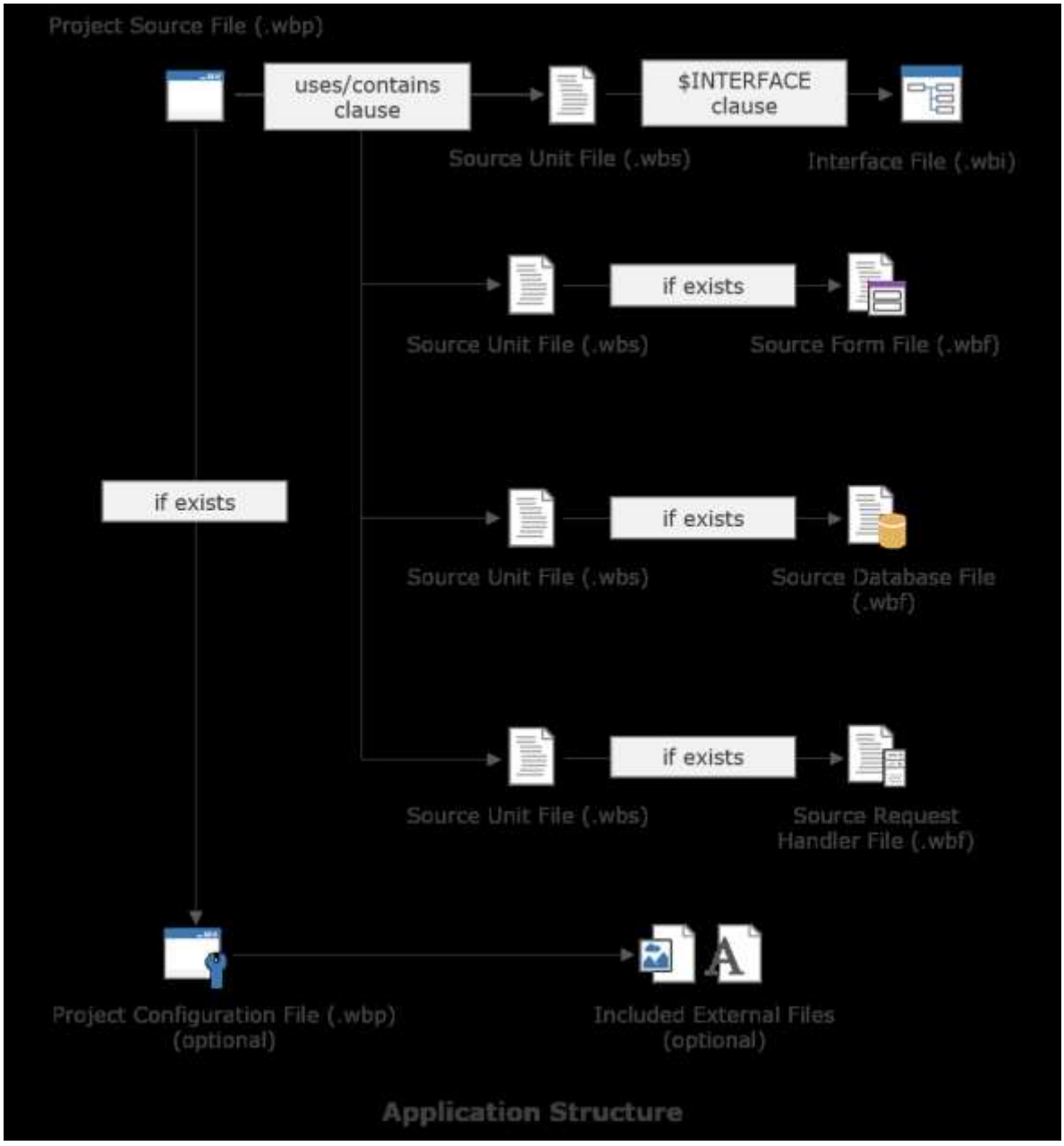
1.4 Application Structure

There are three different types of applications that can be created using the IDE:

Type	Description
Visual Client	Includes access to the form/control components in the component library and allows the developer to visually design and layout the controls in the IDE.
Non-Visual Client	Provides a basic project shell that can be used to create applications that don't automatically use the form/control components in the component library. This is useful when you wish to create a small application that runs in web browsers but accesses the browser APIs and UI elements manually, such as when you have static HTML pages for which you want to provide some dynamic functionality and don't want to use JavaScript.
Server	Includes access to the request handler/non-visual components in the component library and allows the developer to design, code, and debug server-side functionality in the IDE.

All three types of applications are comprised of a project source file (.wbp), a project configuration file (.wbc), and, optionally, one or more source files (.wbs) and form, database, or request handler source files (.wbf). Both database (visual client and server applications) and request handler (server applications) source files use the same format and .wbf file extension as the form files in visual client applications. Also, visual client applications and server applications always contain one main source unit (.wbs) and source form/request handler file (.wbf) that is considered the main entry point for the application. Non-visual client applications do not contain such a unit.

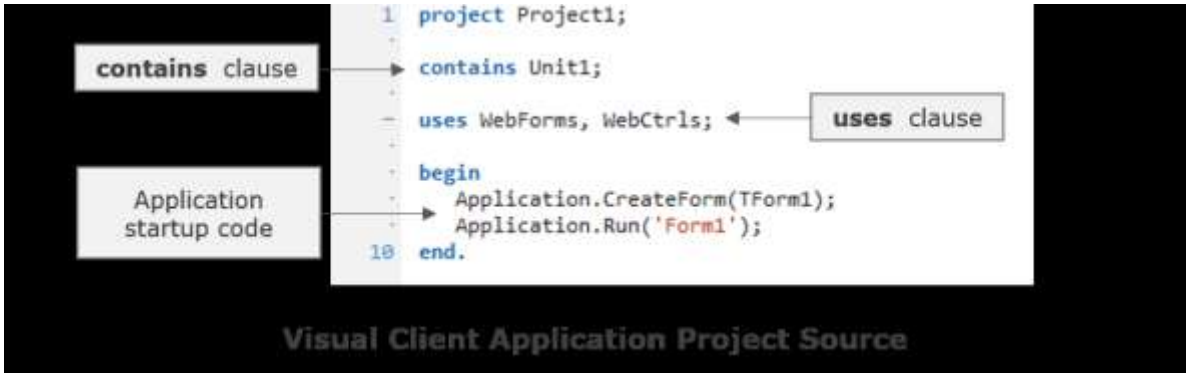
In addition, visual client applications can contain source units that include compiler directives that bundle control interface files with the source code. Many source units in the Elevate Web Builder component library contain such directives.



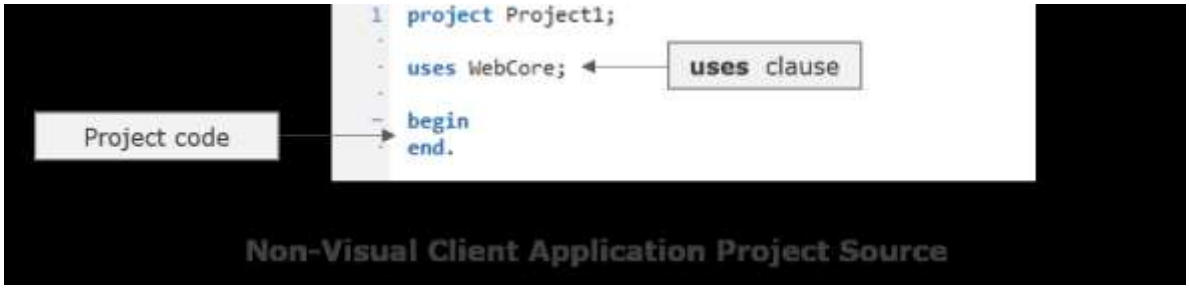
Project Source File

The main difference between visual client projects and non-visual client projects lies in the way that the IDE generates code in the project source file as source units and their associated form/database source files are added and removed from the project. Server applications mimic visual client applications in how the IDE manages the project source file and how source units and their associated database/request handler source files are managed.

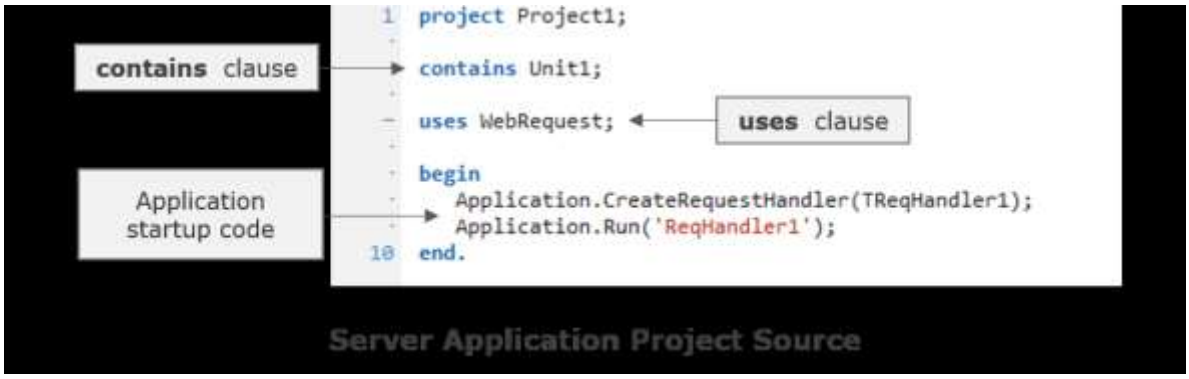
The project source file (.wbp) looks like the following for a visual client application:



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



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



Contains Clause

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

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

Uses Clause

When a project is opened in the IDE, the **uses** clause in the project source file is what determines how the IDE treats the project in terms of application type. There are certain units that must be present in order for the IDE to treat the project as a certain application type, and they are as follows:

Type	Required Unit
Visual Client	WebForms unit
Server	WebRequest unit

Note
Non-visual client applications have neither unit present in the uses clause of the project source file.

Project Configuration File

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

- IDE layout settings (open windows, panel positions/sizes)
- Compilation settings (compiler directives, search paths, output paths, compression)
- External files (JavaScript, images)
- Deployment settings (deployment server/path)

Form, Database, and Request Handler Files

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

Interface Files

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

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

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

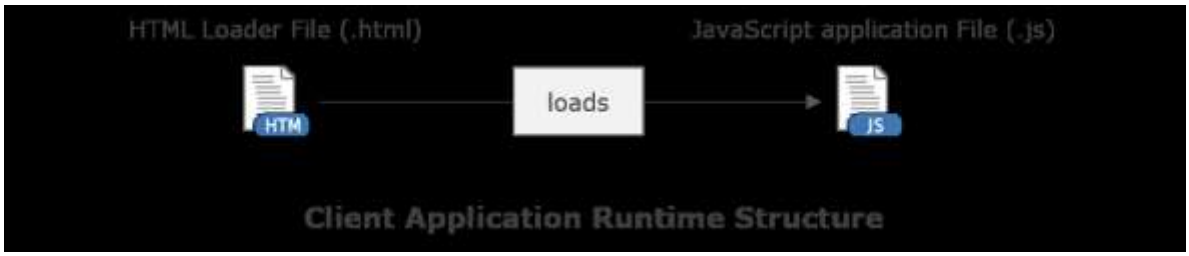
1.5 Building Applications

When you build an application, the compiler uses the project's compiler search paths along with the component library search paths to determine where to look for units (*.wbs) and control interface files (*.wbi). Please see the Modifying Project Options and Modifying IDE Settings topics for more information on modifying these search paths.

The process of loading and executing an application depends upon whether the application is a visual client application, a non-visual client application, or a server application.

Visual and Non-Visual Client Applications

For visual and non-visual client applications, the compiler compiles the application source code (Object Pascal variant) into a 100% JavaScript web application that will run in any modern browser. During the compilation of a visual or non-visual client application, the compiler emits the following files:



For visual client applications, the HTML loader file contains all of the control interface files and form/database files (*.wbf) in special tags in the header of the file. For non-visual client applications, the HTML loader file is created but isn't required to be used and is typically a skeleton that simply loads the application's .js (JavaScript) file.

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

- The HTML loader file loads the application's .js (JavaScript) file, which causes the browser to compile the JavaScript and prepare the execution environment.
- A special JavaScript loader function is called that initializes the application and starts execution.
- Any control interfaces are loaded from the special tags in the HTML loader file.
- For visual client applications, any forms and databases that are designated to be automatically created during application startup are created. During creation, the associated form and database files (*.wbf) are loaded from the special tags in the HTML loader file. Please see the Modifying Project Options topic for more information on designating forms and databases for automatic creation during application startup.
- If the web browser navigates away from the current URL, or if the web browser refreshes the current URL, then a special JavaScript unloader function is called that cleans up all allocated resources and terminates the application.

Client applications are designed to be loaded and then stay loaded until the browser navigates away from the URL that loaded the application. They are not "page-oriented" like many web applications or general web sites.

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

Server Applications

For server applications, the compiler compiles the application source code (Object Pascal variant) into an application file that can be executed by the web server. During the compilation of a server application, the compiler emits a single .wba application file.

The application file contains all of the application source files and any referenced source files, including request handler/database files (*.wbf), in a compressed format.

A server application is loaded by the web server when an incoming web server request is routed to a server application resource that matches the application name assigned to the server application when the application was installed. Once the server application is loaded, the following steps occur:

- The web server loads the .wba application file, uncompresses the source files, and compiles the source code.
- The web server initializes the application and starts execution.
- Any request handlers and databases that are designated to be automatically created during application startup are created. During creation, the associated request handler and database files (*.wbf) are loaded from the application file. Please see the Modifying Project Options topic for more information on designating request handlers and databases for automatic creation during application startup.
- The incoming web server request is passed into the application's main request handler, where the application can handle the request and send a response to the request.
- After the application execution completes, the web server will finalize the application.

1.6 Component Library

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

> **Note**
> The component libraries are the foundation for all design-time visual designers: you cannot work with visual client application projects or server application projects unless the corresponding client or server component library has been successfully loaded and compiled.

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

The component library search paths are a global setting in the IDE and are automatically appended to the compiler search paths during the compilation of projects. Please see the Modifying IDE Settings topic for more information on modifying the component library search paths used by the compiler.

Core Units

There are a set of core units that make up the base functionality for both the client and server component libraries. Some of the core libraries are shared between both, while others are platform-specific and only apply to client or server applications.

Shared Units

The following source units make up the core of the runtime and component library for both client and server applications:

Source Unit	Description
WebCore	This source unit contains the TPersistent class and the TComponent class, the TReader and TWriter JSON persistence classes, the TAbstractList, TObjectList, TStringList, and TCollection list and collection classes, and the TFormatSettings date/time and number formatting class. In addition, many of the runtime functions and procedures that aren't intrinsic to the runtime are implemented in this source unit.
WebData	This source unit contains the TDatabase and TDataSet components.
WebHTTP	This source unit contains the TServerRequest component.

Client Units

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

Source Unit	Description

WebDOM	This source unit contains all external declarations for the web browser DOM (Document Object Model) classes, functions, procedures, and global variables. You can use the classes and functionality in this source unit to manipulate the browser DOM directly at run-time.
WebUI	This source unit contains the interface manager and the abstract UI layer that is used in both the design-time environment and the run-time environment.
WebCtrls	This source unit contains the TControl component, which is the base control for all visual controls, as well as other controls that inherit from the TControl component to provide base functionality for scrollable, bindable, and input controls.
WebForms	This source unit is the base unit for visual client applications and contains the TApplication, TSurface, TForm, TDialog, and TFrame components.

Server Units

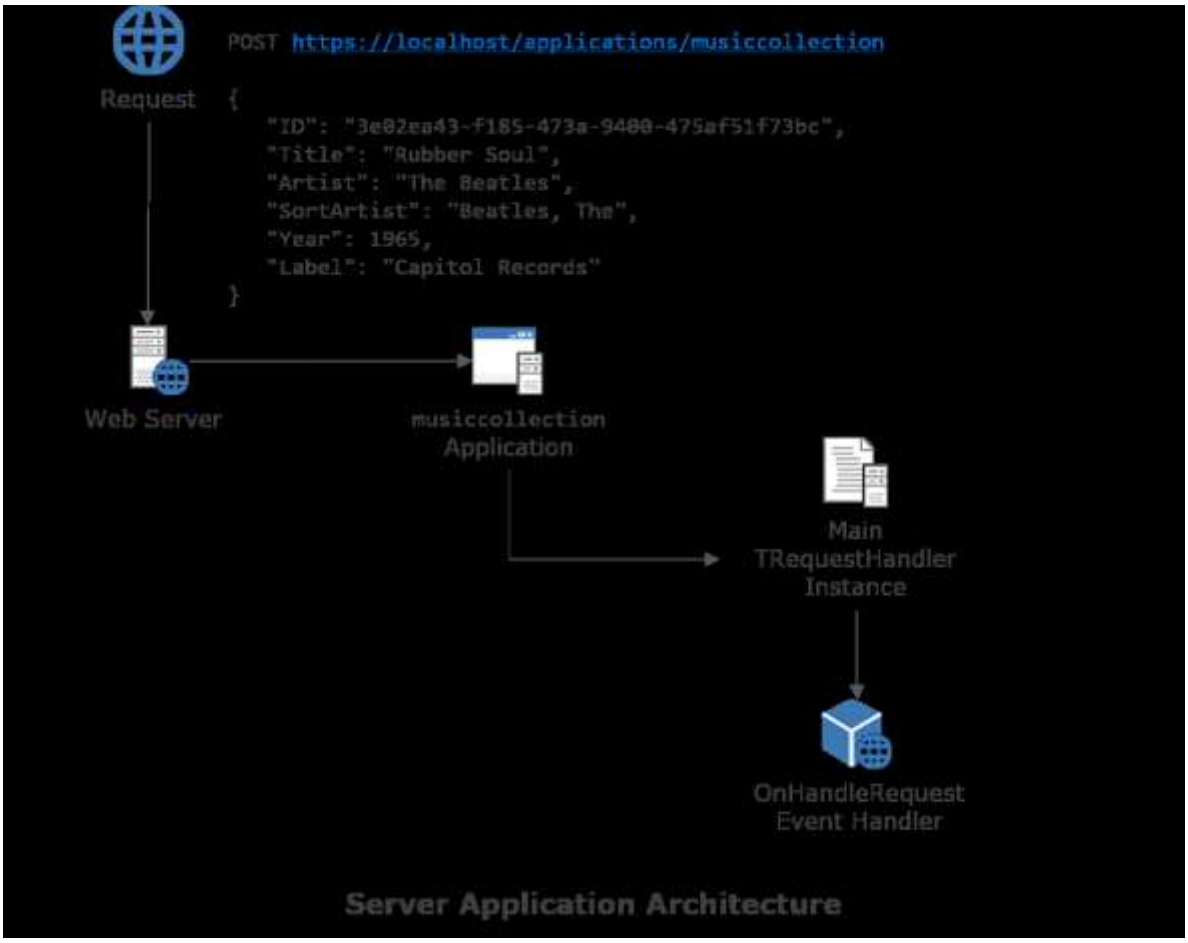
The following source units make up the core of the runtime and component library for server applications:

Source Unit	Description
WebSrvr	This source unit contains all external declarations for the web server classes, functions, procedures, and global variables. This functionality is typically used in other source units in the component library, but certain classes, such as the TStream, TRasterImage, TWebServerRequest, and TWebServerSession classes, are used directly in applications.
WebRequest	This source unit is the base unit for server applications and contains the TApplication and TRequestHandler components.

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

1.7 Web Server Applications

The interface of an Elevate Web Builder server application has the following structure:



A server application uses a global instance of the TWebServerRequestManager class called RequestManager to manage all aspects of the incoming web server request. The TWebServerRequestManager class and all related classes, types, and functions/procedures are declared in the WebSrvr unit included with the standard component library. The incoming request is represented by an instance of the TWebServerRequest class, which is itself a descendant of the base THTTPServerRequest class.

Core Application Components

The server application functionality contains several core components, all residing in the WebRequest unit in the standard component library.

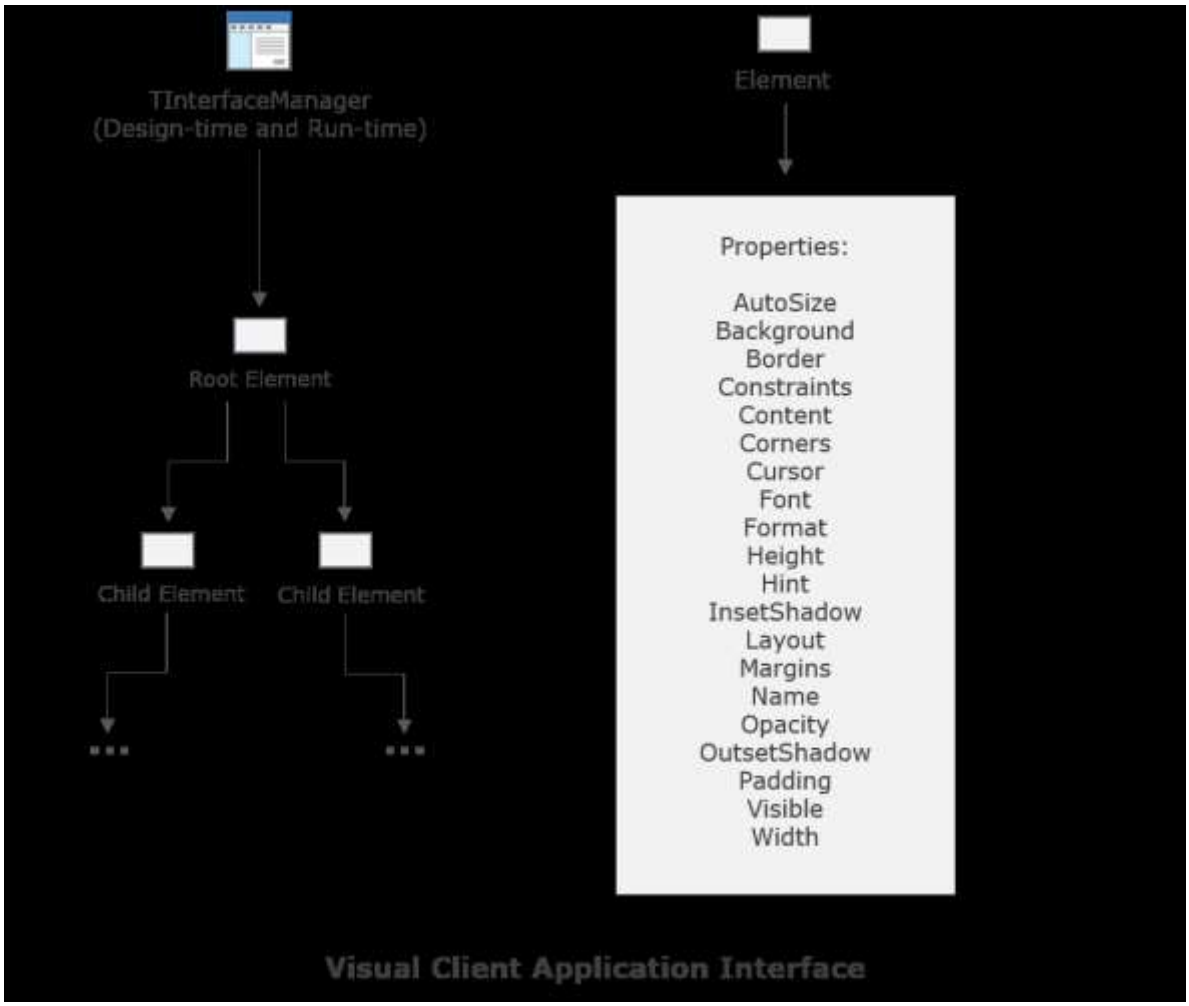
TApplication
An instance of the TApplication component is automatically created when the WebRequest unit is initialized, and is available as the global Application variable (also in the WebRequest unit). The TApplication component allows the developer to manage various aspects of the application such as the application title and version.

TRequestHandler
The TRequestHandler component encapsulates a request handler in a server application. Request handlers are a non-visual container in which other non-visual components can reside. Request handlers can be designated as auto-create designer instances in a project and the IDE will automatically add the appropriate code to the program source of the project for creating the request handlers at application startup. Please see the Modifying Project

Options topic for more information on designating request handlers as auto-create designer instances. The TRequestHandler class can be found in the WebRequest unit.

1.8 Visual Client Applications

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



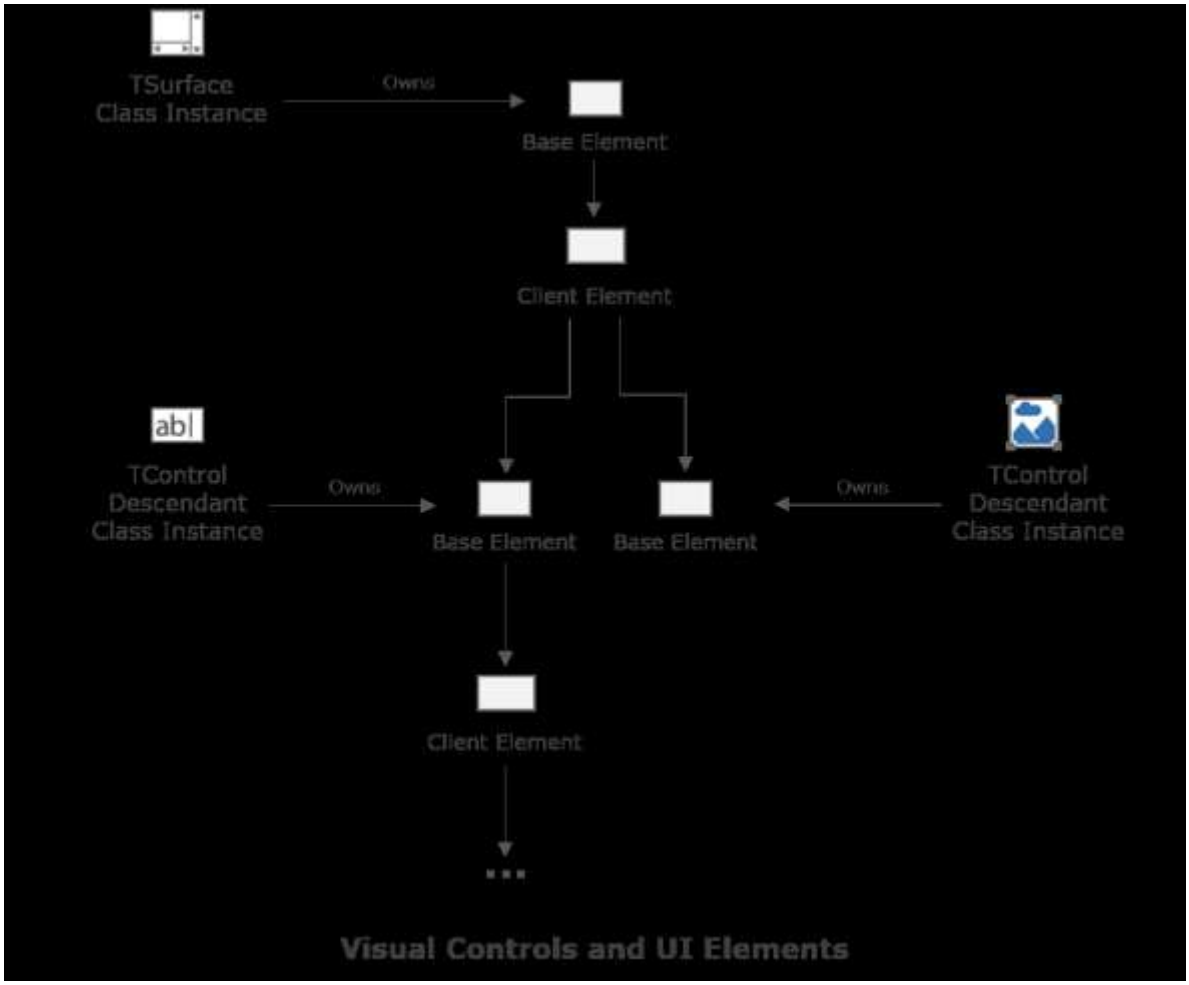
A visual client application uses a global instance of the TInterfaceManager class called InterfaceManager to manage all aspects of the user interface. The TInterfaceManager class and all related classes, types, and functions/procedures are declared in the WebUI unit included with the standard component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

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

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

Controls and Interface Elements

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



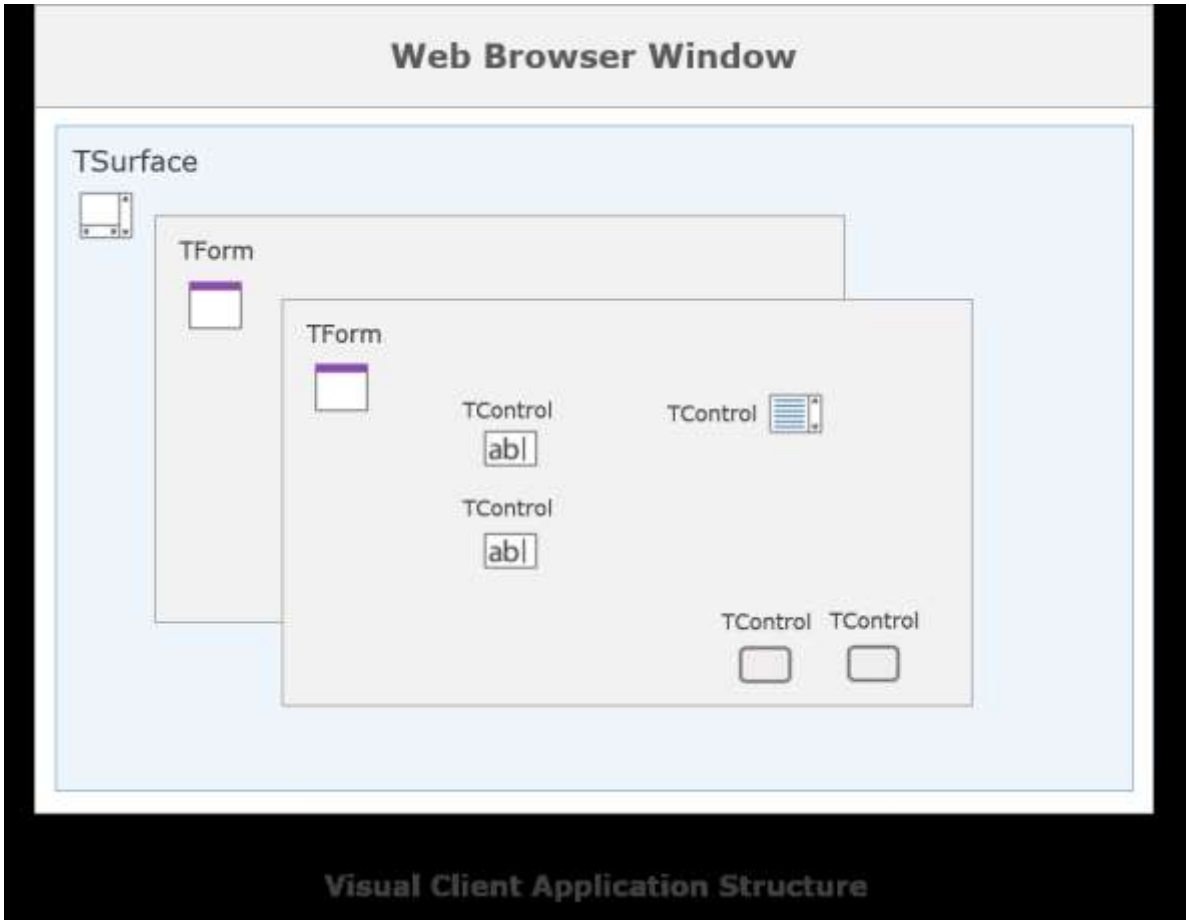
Many controls never use more than the base element, while others such as grid controls require many elements in order to provide the necessary functionality in the control.

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

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

Core Application Components

A visual client application has the following structure in the web browser:



Visual Client Application Structure

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

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

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

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

TForm
The TForm component encapsulates a form in a visual client application. Forms are the container controls in which all other visual controls reside. Forms can be designated as auto-create designer instances in a project and the IDE will automatically add the appropriate code to the program source of the project for creating the forms at application startup. Please see the Modifying Project Options topic for more information on designating forms as auto-create designer instances. The TForm class can be found in the WebForms unit.

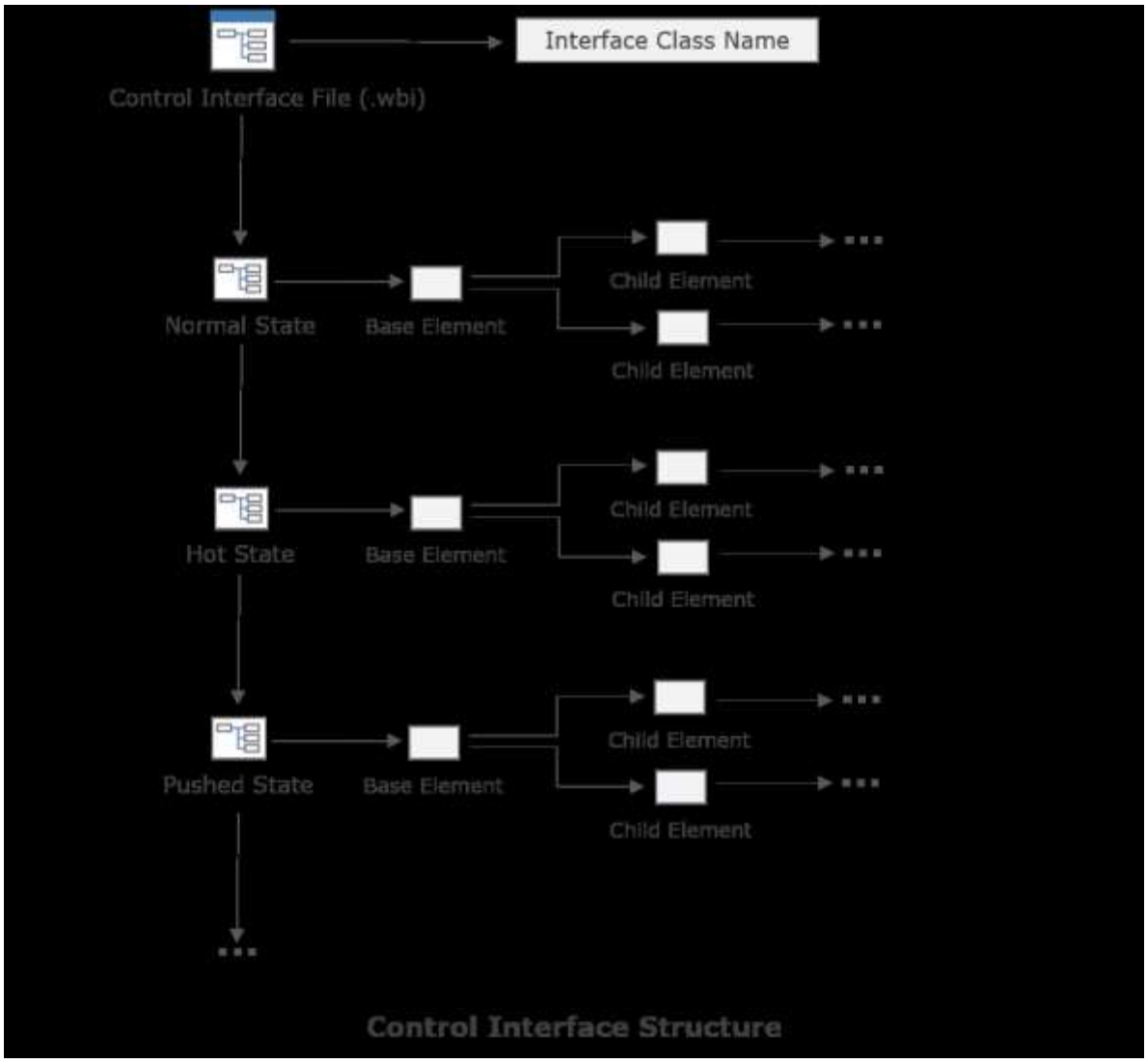
TDialog
The TDialog component encapsulates a dialog in a visual client application. Dialogs differ from normal forms in that

they contain additional interface elements such as a caption bar and a close button. Also, dialogs are normally shown modally using the TFormControl ShowModal method, although this is not a requirement. The TDialog class can also be found in the WebForms unit.

TFrame
The TFrame component encapsulates a frame in a visual client application. Frames differ from normal forms in that they do not have any scrolling functionality and can be auto-sized (height, width, or both). Frames are, therefore, lightweight and more useful in situations where you wish to embed many instances of a given frame in a parent container control, as is the case with the Music Collection and Photo Album example visual client applications that ship with Elevate Web Builder. The TFrame class can also be found in the WebForms unit.

1.9 Control Interfaces

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



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

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

TControl Interface Functionality

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

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

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

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

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

How Control Interfaces are Applied to a Control

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

- The interface class name
- A specified interface state
- The interface element names

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

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

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

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

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

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

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

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

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

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

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

Customizing Control Interfaces

Elevate Web Builder ships with two complete sets of standard control interfaces located in the \interfaces subdirectory under the main installation directory:

Control Interface Set	Description
Default	This is the default set of control interfaces and is intended to offer a Windows "Metro" look similar to Windows 8 and higher. It uses the Seqoe UI font and font sizes that are big enough for most touch environments.
Desktop	This is a set of control interfaces intended to be used primarily for desktop applications. It also uses the Segoe UI font, but with smaller font sizes and narrower borders.

Because the compiler uses the standard project and component library search paths to find both source units and control interfaces, you can make copies of the standard control interfaces, place them in a new directory, and then modify the project's compiler search paths so that the new directory is included. Please see the Modifying Project Options topic for more information on modifying the project's compiler search paths. After that point, any modifications to the control interfaces in this new directory will be used instead of the standard control interfaces. Please see the Opening a Control Interface topic for more information on how to open and modify existing control interfaces using the control interface designer.

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

Note
The Music Collection, Photo Album, and Contact Form example applications use customized control interfaces to offer unique interfaces for each application. You can open these example applications in the IDE to see what effect customized control interfaces have upon the look and feel of the applications.

1.10 Icon Library

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

Supported Icon Types

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

Icon Fonts

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

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

Font Awesome

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

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

Defining Icons

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

Image Icons

- The base element for each icon (state) **must** be named "Icon".
- The "Icon" element's ApplyProperties Background property should be set to True.

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

Font Icons

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

Defining Animated Icons

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

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

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

Loading Icons from Code

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

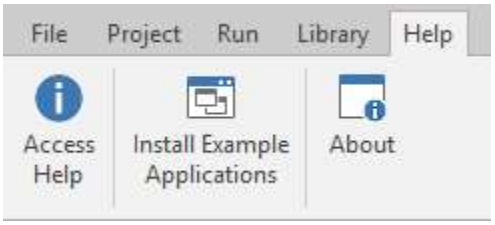
1.11 Accessing Help

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

Accessing the Online Manual

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

- Click on the **Help** tab on the main menu. The Help menu will appear:

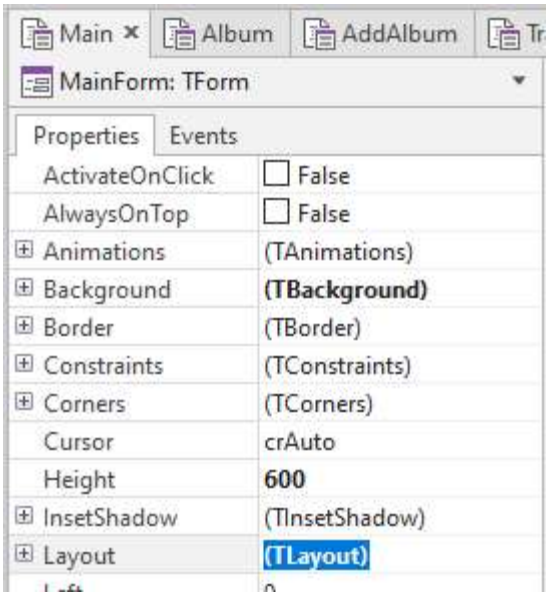


- Click on the **Access Help** button on the Help menu. This will cause the help browser page to open in the IDE.

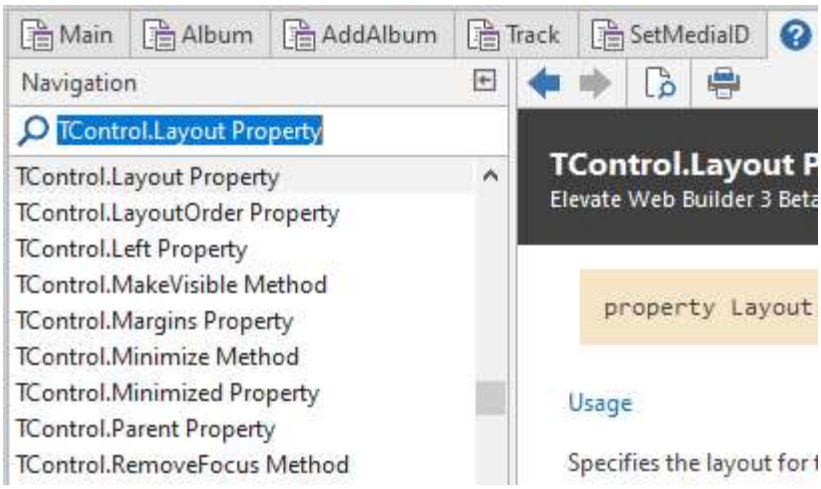
Context-Sensitive Help in the Component Inspector

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

- Click on the desired property in the component inspector and press the F1 key.



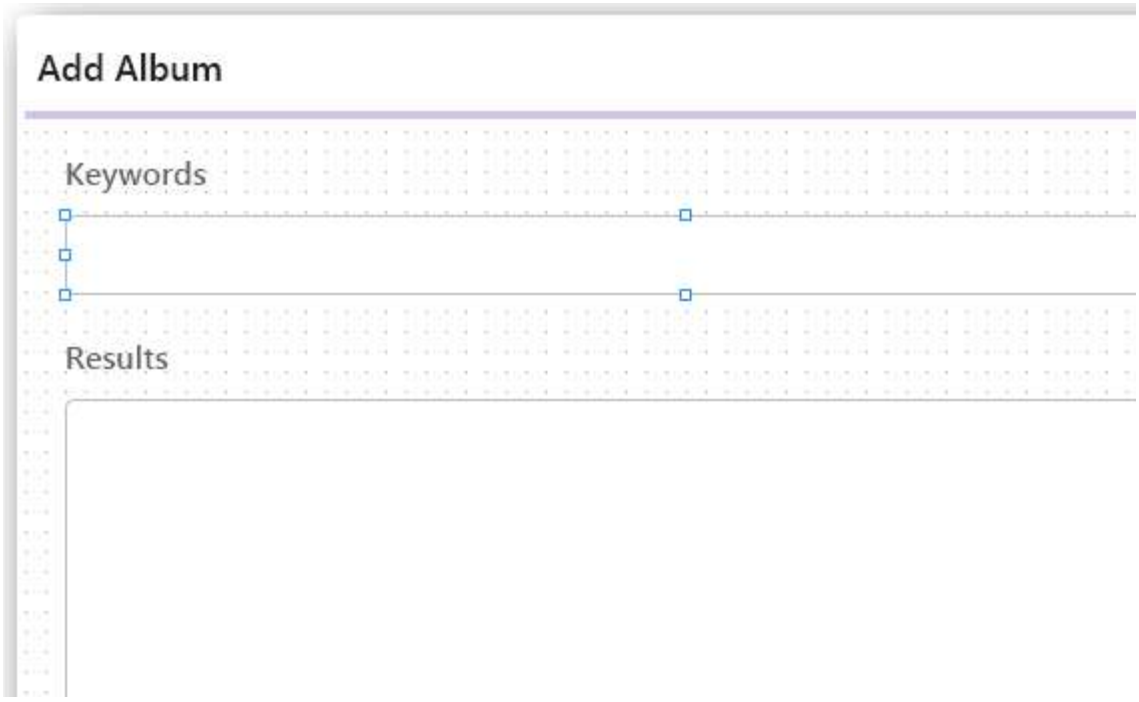
- The help browser will open with a listing of all of the topics that match the property. The topic that corresponds to the first matching property will be displayed in the help browser window. If there is only one matching topic, then the listing will not be shown and you'll see only the matching property in the help browser window.



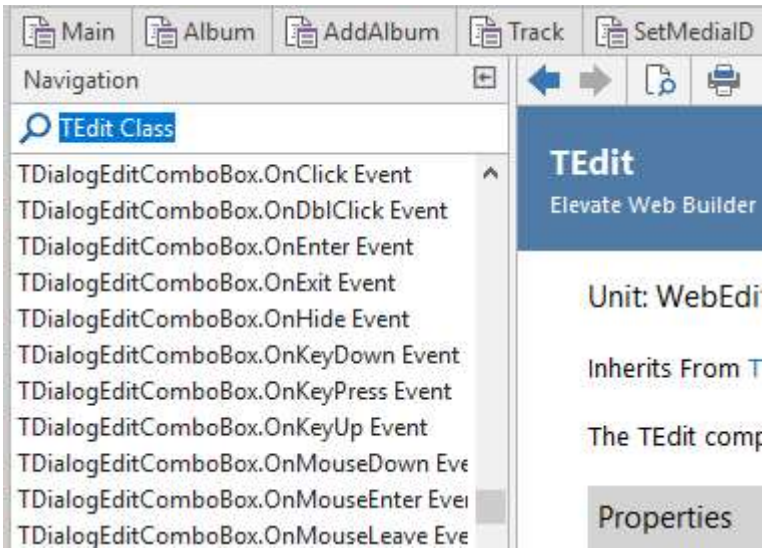
Context-Sensitive Help in the Form and Database Designers

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

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



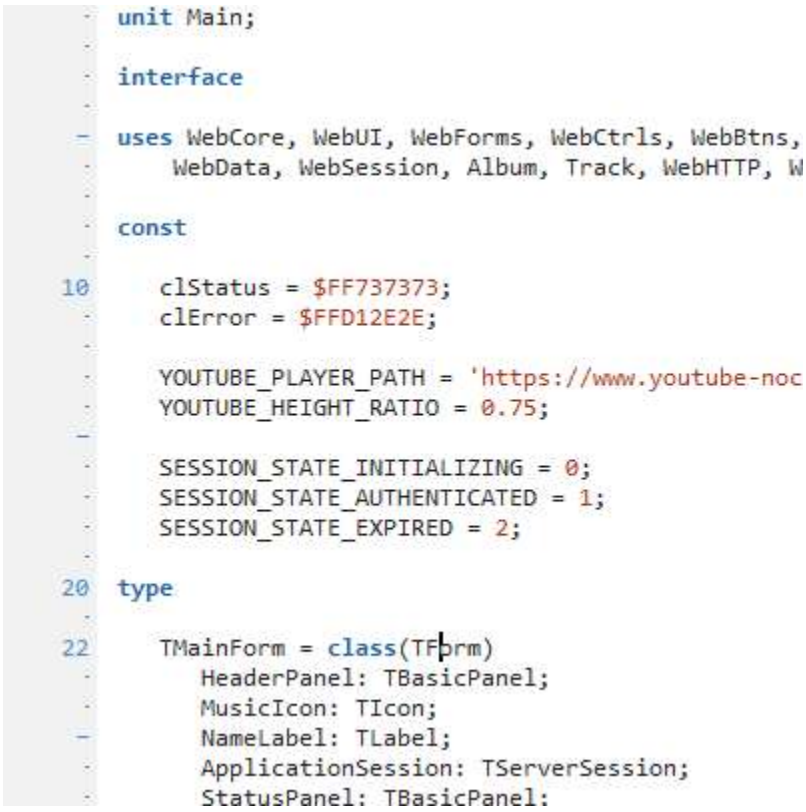
- The help browser will open with a listing of all of the topics that match the control or component. The topic that corresponds to the first matching control or component will be displayed in the help browser window. If there is only one matching topic, then the listing will not be shown and you'll see only the matching control or component in the help browser window.



Context-Sensitive Help in the Code Editor

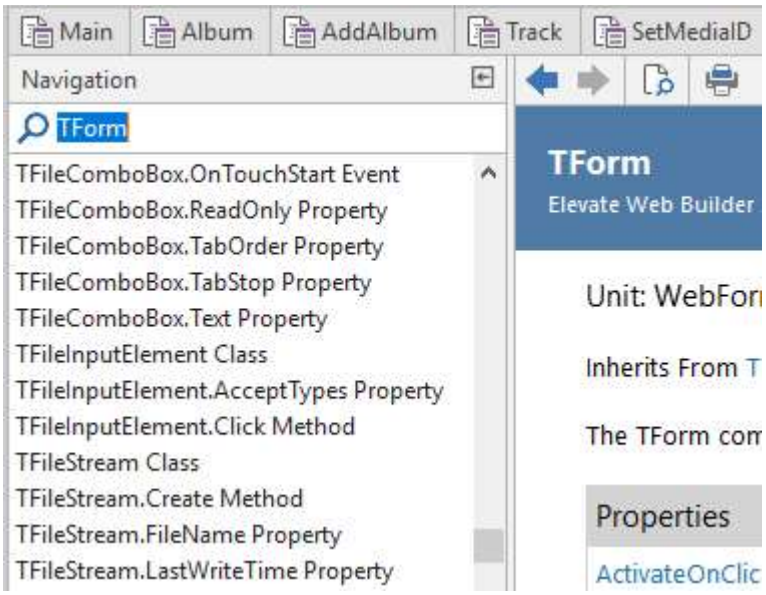
Use the following steps in order to obtain context sensitive help in the code editor:

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



```
unit Main;

interface

uses WebCore, WebUI, WebForms, WebCtrls, WebBtns,
    WebData, WebSession, Album, Track, WebHTTP, W

const

   clStatus = $FF737373;
   clError = $FFD12E2E;

   YOUTUBE_PLAYER_PATH = 'https://www.youtube-noc
   YOUTUBE_HEIGHT_RATIO = 0.75;

   SESSION_STATE_INITIALIZING = 0;
   SESSION_STATE_AUTHENTICATED = 1;
   SESSION_STATE_EXPIRED = 2;

type

   TMainForm = class(TForm)
      HeaderPanel: TBasicPanel;
      MusicIcon: TIcon;
      NameLabel: TLabel;
      ApplicationSession: TServerSession;
      StatusPanel: TBasicPanel;
```

- The help browser will open with a listing of all of the topics that match the keyword or identifier. The topic that corresponds to the first matching keyword or identifier will be displayed in the help browser window. If there is only one matching topic, then the listing will not be shown and you'll see only the matching keyword or identifier in the help browser window.



Using the Help Browser

You can choose to navigate the help using the table of contents or the help index by selecting the appropriate tab in the bottom left-hand corner of the help browser:

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

The navigation buttons are:

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

In addition you can use the following toolbar buttons:

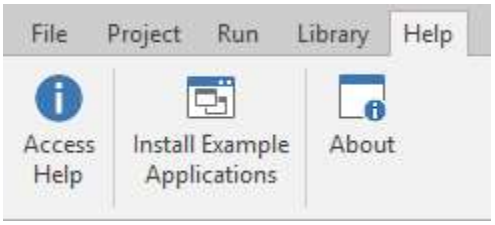
 Searches for text within the current topic.

 Prints the current topic to the desired output device.

1.12 Example Applications

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

- Click on the **Help** tab on the main menu. The Help menu will appear:



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

```
My Documents\Elevate Web Builder 3\Projects
```

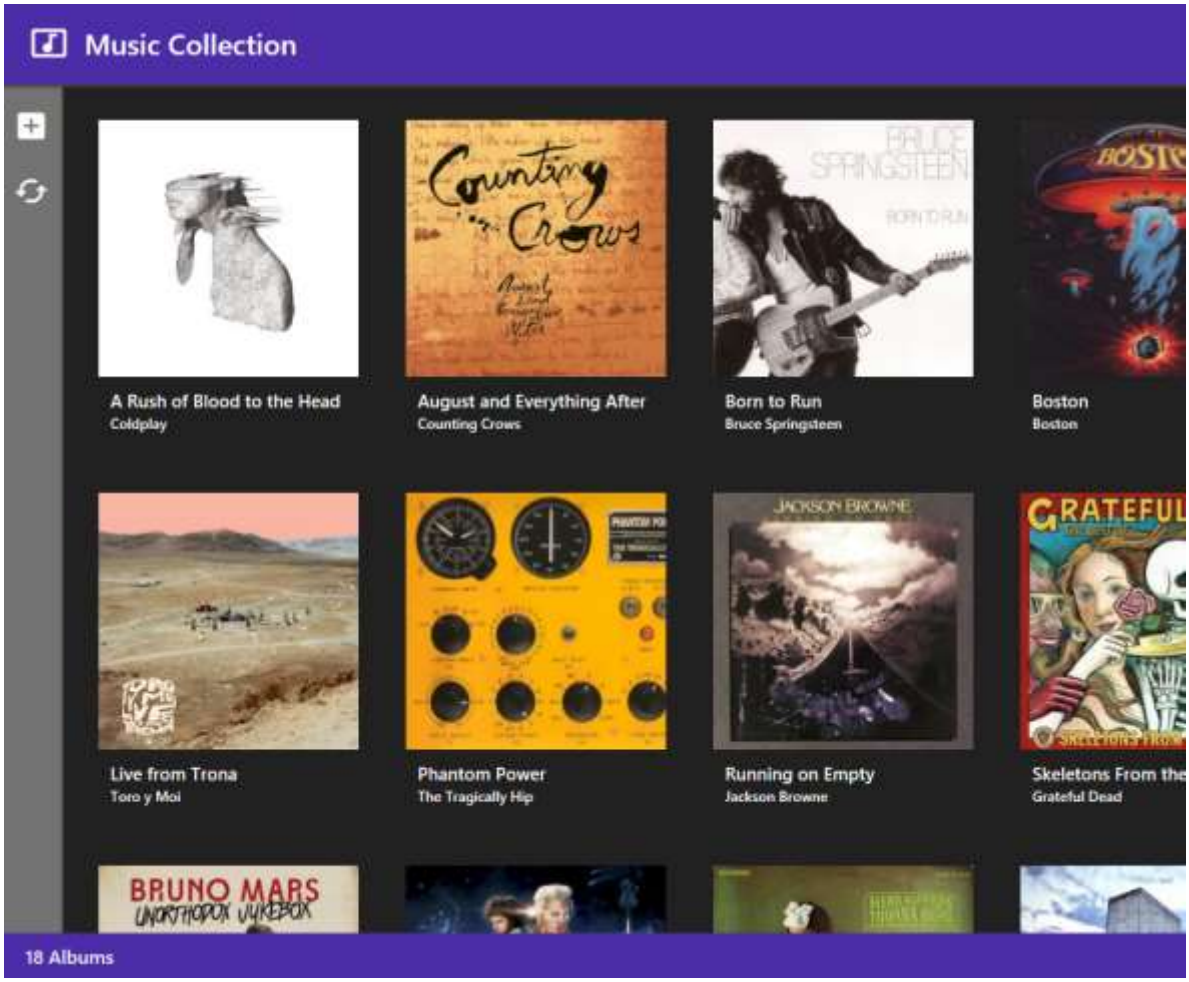
All of the example applications require that the internal web server in the IDE is started and running. If the internal web server is not running in the IDE, you will see errors when trying to run the example applications in the IDE (or from any browser). Also, do not attempt to run any of the example applications directly from the file system in the IDE (or from any browser), doing so will result in errors. Please see the Running and Debugging a Project topic for more information on running projects.

Additionally, all of the example applications use customized control interfaces for the client application project portion of the example. These customized control interfaces can be found in the \Client\Interfaces subdirectory under the base example directory. Please see the Control Interfaces topic for more information on how control interfaces work and how to customize them on a per-project basis.

Finally, all of the example applications use the Google Material Design icon font for the icons in each client application project. The TrueType and WOFF web font versions of this icon font are automatically installed during the product installation, and the WOFF web font version of this icon font is set up for each client application project when the example applications are installed.

Music Collection Example

Scope: Client and Server



After installation, the Music Collection example application projects will be located in the My Documents\Elevate Web Builder 3\Projects\MusicCollection directory. The Music Collection example application includes both a client application and a server application. The client application project will be located in the \Client directory under the base example directory and the server application project will be located in the \Server directory under the base example directory.

Note
The server application is automatically built and deployed to the internal web server in the IDE when the example applications are installed. Therefore, you will only need to open and run the client application in order to see the application live in the IDE. However, pre-defined requests are created in the Request Manager in the IDE when the example applications are installed, so you can also execute the server applications directly in the IDE by using these pre-defined requests.

The Music Collection example illustrates the following server application techniques:

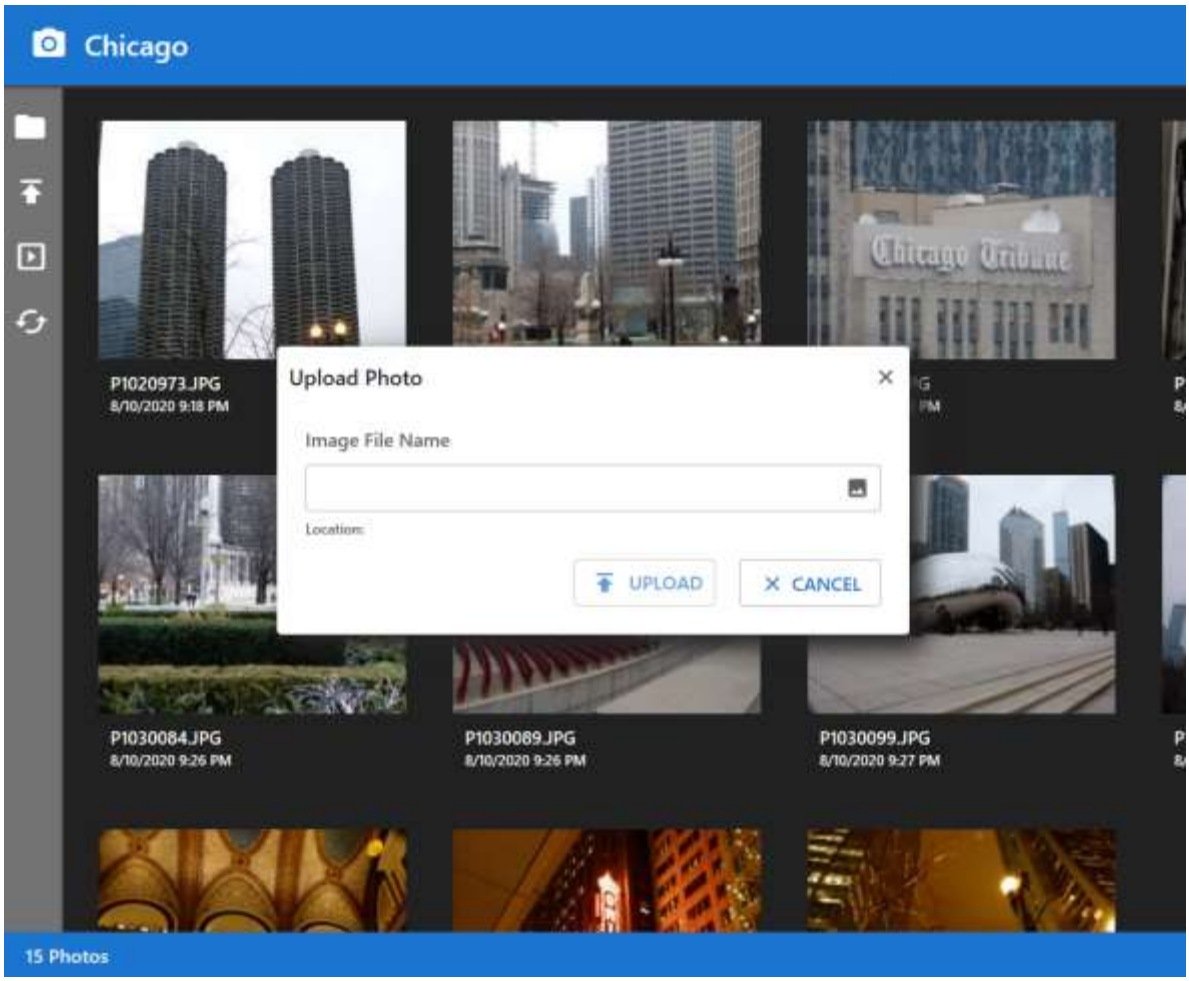
- How to handle requests for a REST API using the TRequestHandler component
- How to execute requests to an external REST API (the MusicBrainz music database JSON API) using the TServerRequest component and load the JSON responses using custom TPersistent classes
- How to use the TDatabase component to perform transactions and the TDataSet component to perform dataset searches

The Music Collection example illustrates the following client application techniques:

- How to use the TServerSession component to authenticate against a web server
- How to use the TServerRequestQueue component to execute requests to the REST API exposed by the MusicCollection server application, including handling authentication failures due to session expiration and retries of requests
- How to use the TFrame control to embed custom TFrame instances in a parent control using a scrollable flow layout
- How to define animations for controls using the TControl Animations property
- How to use the TBrowser control to display external content like YouTube videos
- How to use the TDataSet component to load data for display

Photo Album Example

Scope: Client and Server



After installation, the Photo Album example application projects will be located in the My Documents\Elevate Web Builder 3\Projects\PhotoAlbum directory. The Photo Album example application includes both a client application and a server application. The client application project will be located in the \Client directory under the base example directory and the server application project will be located in the \Server directory under the base example directory.

Note
The server application is automatically built and deployed to the internal web server in the IDE when the example applications are installed. Therefore, you will only need to open and run the client application in order to see the application live in the IDE. However, pre-defined requests are created in the Request Manager in the IDE when the example applications are installed, so you can also execute the server applications directly in the IDE by using these pre-defined requests.

The Photo Album example illustrates the following server application techniques:

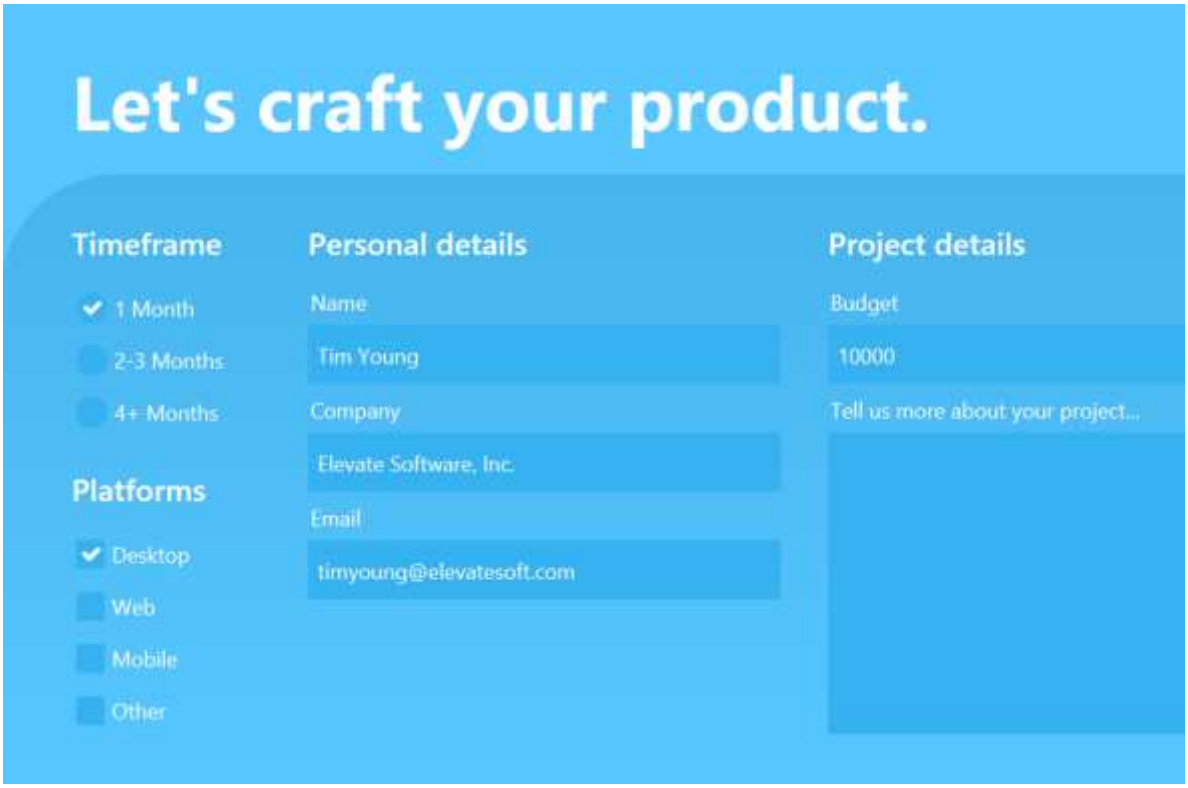
- How to handle requests for a REST API using the TRequestHandler component
- How to use the TRasterImage class to load an image from a TStream instance and resize it to create a thumbnail version of the image
- How to use the TDatabase component to perform transactions and the TDataSet component to perform dataset searches

The Photo Album example illustrates the following client application techniques:

- How to use the TServerSession component to authenticate against a web server
- How to use the TServerRequestQueue component to execute requests to the REST API exposed by the PhotoAlbum server application, including handling authentication failures due to session expiration and retries of requests
- How to assign the THTMLForm control to TServerRequest component instances in order to allow for finer-grained control over HTML file uploads
- How to use the TFrame control to embed custom TFrame instances in a parent control using a scrollable flow layout
- How to define animations for controls using the TControl Animations property
- How to use the TDataSet component to load data for display

Contact Form Example

Scope: Client and Server (Native Server Module)



After installation, the Contact Form example application projects will be located in the My Documents\Elevate Web Builder 3\Projects\ContactForm directory. The Contact Form example application includes both a client application and a native server module. The client application project will be located in the \Client directory under the base example directory and the native server module project will be located in the \Module directory under the base example directory.

Note
The native server module is pre-built and digitally-signed by Elevate Software when the Elevate Web Builder installation is built. It is also deployed to the internal web server in the IDE when the example applications are installed. Therefore, you will only need to open and run the client application in order to see the application live in the IDE. However, the native server modules are created and built using the Embarcadero RAD Studio (Delphi) product. If you want to modify or re-compile a native server module project, then you must download and install the Elevate Web Builder Modules installation available for Embarcadero RAD Studio (Delphi) XE2 or higher. Please see the Creating Web Server Modules topic for more information on downloading the Elevate Web Builder Modules installation.

The Contact Form example illustrates the following native server module techniques:

- How to initialize a database engine during the initialization of the native server module (this is important for both the DBISAM and ElevateDB database engines)
- How to use the TEWBDatabaseAdapter and TEWBDataSetAdapter components to expose a database REST API from a native server module

The Contact Form example illustrates the following client application techniques:

- How to use the TDatabase component to transparently authenticate against a web server and how to use manual transactions against the REST database API exposed by the ContactForm native server module

- How TDataSet component to insert a new row in a dataset using the REST database API exposed by the ContactForm native server module
- How to define animations for controls using the TControl Animations property

This page intentionally left blank

Chapter 2
Using the Web Server

2.1 Configuring the Web Server

Elevate Web Builder includes a web server that can be used for serving up static content, including client browser applications. The web server is also an application server that provides a server application execution context as well as authentication, administration, database, and logging APIs and functionality. This web server is made available in several different versions.

Internal Web Server

For ease of use, the IDE includes an embedded version of the web server for development. By default, this web server is called **Internal** and is automatically set up during IDE startup with a specific configuration for use with the **localhost** loopback host and the standard HTTP/HTTPS ports 80 and 443. This configuration is stored in the ewbide.ini IDE settings file located here:

```
C:\Users\<UserName>\AppData\Local\Elevate Software\Elevate Web Builder
      3\ewbide.ini
```

where <UserName> is the name of the user account under which the IDE is being run. The Internal web server configuration is stored under the **Server_Internal** section name in the ewbide.ini file.

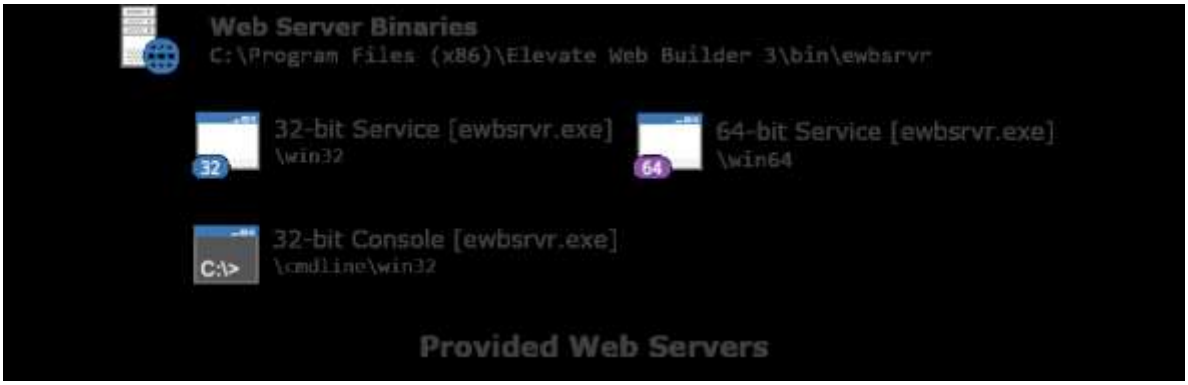
Note
This information is simply for reference, and you do not need to manually modify these settings in the ewbide.ini settings file. Instead, you can simply use the Server Manager to interactively modify the ports and other settings for the Internal web server. Please see the Using the Server Manager topic for more information on how to edit the settings for the Internal web server.

External Web Servers

In addition to the embedded version of the web server, two other external versions of the web server are provided with Elevate Web Builder:

- A Windows service available as 32-bit and 64-bit executables for production
- A Windows console application available as a 32-bit executable for development

The executables for these external web servers are installed in the following locations:



These external web servers use an .ini file for storing their configuration in the following location:

```
C:\ProgramData\Elevate Software\Elevate Web Builder 3 Web Server
```

When any external web server is first started and the .ini configuration file does not exist in the above location, the configuration file will be created automatically with default values for all configuration entries.

The name of the .ini configuration file is determined by the name of the web server executable. For example, for the ewbsrvr.exe Windows service, the name of the .ini file would be ewbsrvr.ini.

Note
If the web server finds an .ini with the proper name in the same directory as the server executable, it will use it instead of the .ini file in the common ProgramData directory for Windows.

By default, all of the configuration entries in the web server .ini configuration file are stored under a section called "Server". However, when running multiple web server instances, each section will have the following format:

```
[Server_<Server Name>]
```

where <Server Name> is the name of the web server instance. Please see the Multiple Server Instances topic for more information.

Server Name and Description

The web server can be given a name and description that are used for a few different purposes:

- The server name is used to identify the service when running the web server as a Windows service.
- The server description is used to describe the service when running the web server as a Windows service, and is used to identify the server via the Server header in the response to each HTTP request.

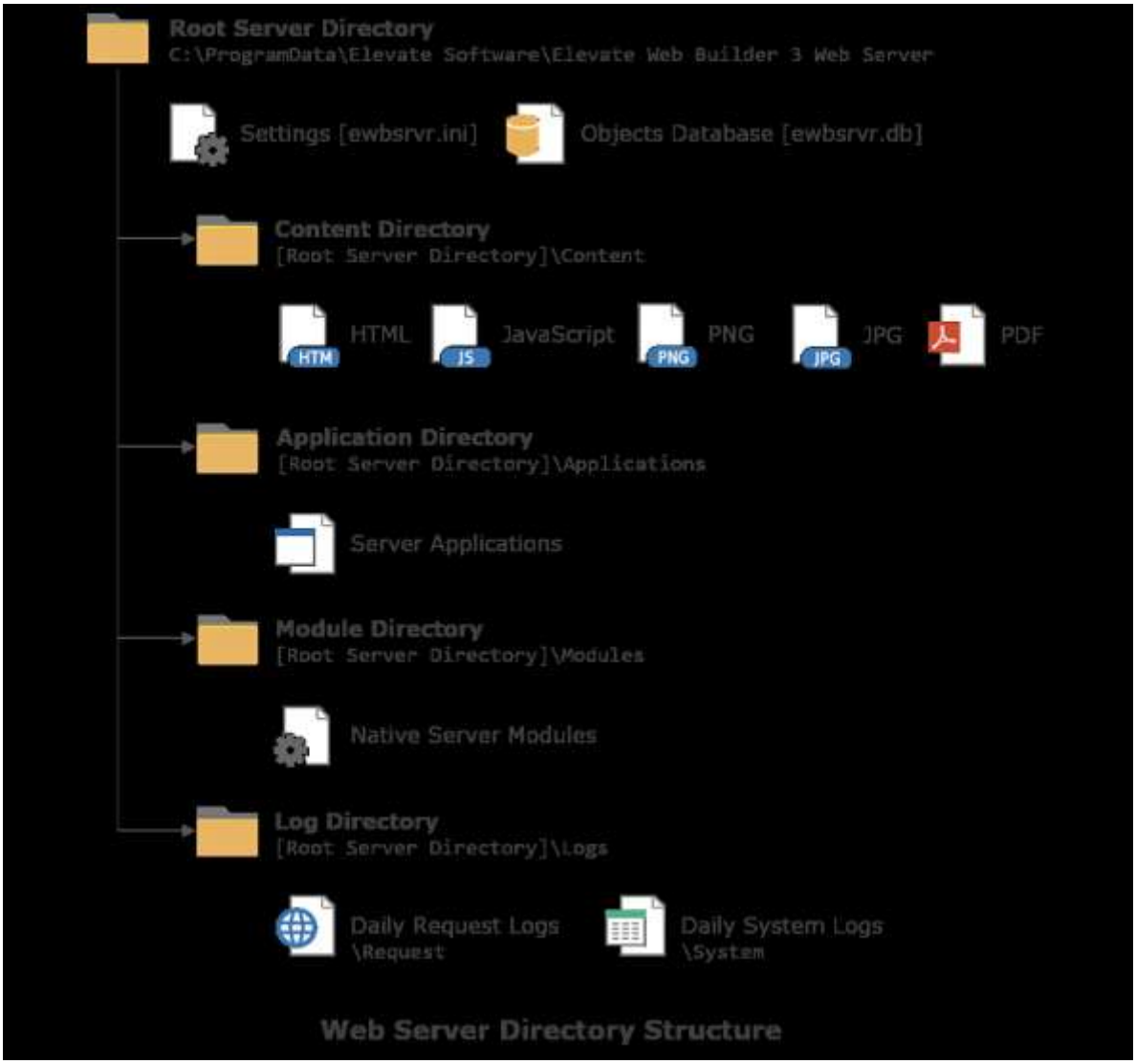
The configuration entries for the web server name and description are as follows:

Configuration Entry	Description

Server Name	Identifies the web server. The default value is "EWBSRVR". **Warning** If using multiple web server instances, then each server instance would occupy a different section in the .ini file and need to have a different Server Name value. Failing to ensure a unique server name could result in Windows service name conflicts and other issues.
Server Description	Specifies a more descriptive name for the web server. The default value is "Elevate Web Builder 3 Web Server".

Directory Structure

The default directory structure used by these external web servers looks like this:



The configuration entries for the server directories are as follows:

Configuration Entry	Description

Content Directory	Specifies the directory that the web server should use as the base directory for all static content, such as HTML files, JavaScript source files, image files, etc. The default value is "content".
Default Document	Specifies the default static HTML file to use if the root URL path is referenced without specifying a file name in a HEAD or GET HTTP request. The default value is blank ("").
Application Directory	Specifies the directory that the web server should use as the base directory for the deployment of server applications. The default value is "applications".
Module Directory	Specifies the directory that the web server should use as the base directory for the deployment of native server modules. The default value is "modules".
Log Directory	Specifies the directory that the web server should use as the base directory for the storage of both the daily HTTP request logs (CSV) and the daily system logs (JSON). The default value is "logs".
Request Logging	Specifies whether HTTP request logging is enabled (the default is 1).
Logging	Specifies whether system logging is enabled (the default is 1).

Server Objects Database

Because the web server also acts like an application server, it requires a database for the storage of information about various server objects, including privileges, roles, users, databases, server applications, and native server modules. This database is called ewbsrvr.db and is, by default, stored here:

```
C:\ProgramData\Elevate Software\Elevate Web Builder 3 Web Server
```

The web server database uses a custom binary format that implements object versioning, allowing the various server objects to be updated in real time without requiring the web server to be restarted. This binary format is compressed when stored on disk.

The web server stores each user password in the database as the hash of a cryptographically-secure random value, called a "salt", plus the password.

All database writes are atomic: writes are first durably written to a transaction log file and then the database is updated. The web server will automatically recover from a database update failure during startup.

The configuration entries for the web server database are as follows:

Configuration Entry	Description
Database Directory	Specifies where the ewbsrvr.db web server database is stored. The default value is "C:\ProgramData\Elevate Software\Elevate Web Builder 3 Web Server".
Max Database Write Delay	Specifies how long, in seconds, the web server waits after an update before writing the updates to the database. The default value is 2 seconds.
Password Salt Length	Specifies the length of the password salt value. The default value is 16 characters.
Password Hash Type	Specifies how the salt+password hash is generated, with 0 being an SHA-256 hash and 1 being an SHA-512 hash. The default value is 0.

Startup

By default, the web server will bind to all available IP addresses on all available local network interfaces, and will bind to two ports: the insecure HTTP port 80 and the secure HTTPS port 443.

> **Note**
> Both ports must be available for the web server to be able to successfully start.

The web server can be configured so that all requests to the insecure HTTP port are automatically redirected to the secure HTTPS port.

The web server always uses long-lived connections and does not automatically drop the connection after a request. The default connection timeout is 30 seconds. Connections are serviced using a thread-per-connection model with threads provided on-demand from a thread cache with a default size of 128 threads. The default maximum of 2048 threads prevents excessive load from causing the web server to exceed the limits of the host operating system.

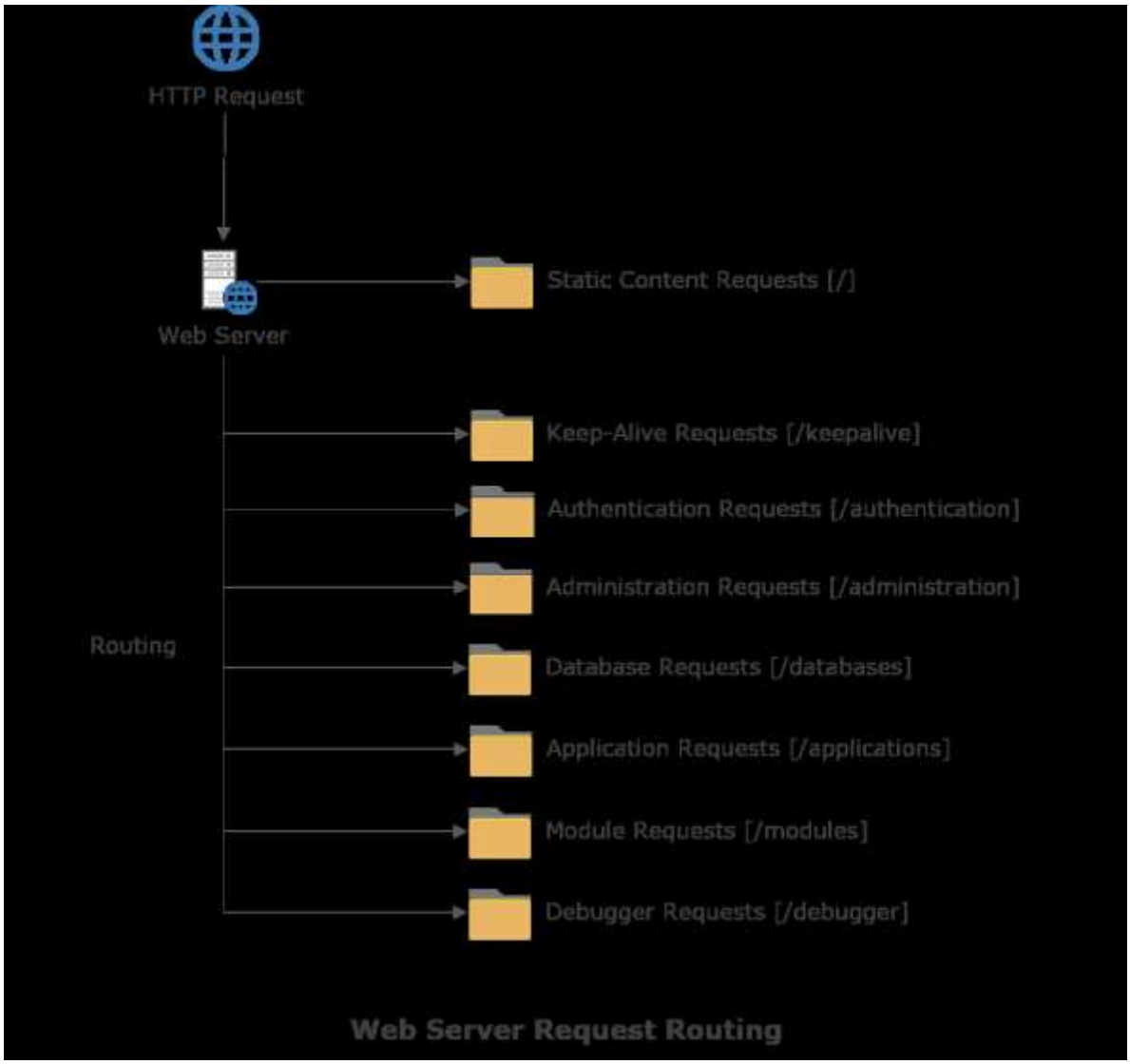
The configuration entries for the web server startup are as follows:

Configuration Entry	Description
IP Address	Specifies the IP address that the web server should bind to. The default value is blank, or all local network interface addresses.
Port	Specifies the insecure HTTP port that the web server should bind to. The default value is 80.
Secure Port	Specifies the secure HTTPS port that the web server should bind to. The default value is 443.
Redirect to Secure	Specifies whether the web server should ensure that any requests to the HTTP port are redirected to the HTTPS port with a 301 HTTP status code (Moved permanently), with 1 being Yes and 0 being No. The default value is 0.
Timeout	Specifies how long, in seconds, the web server will keep a connection open while waiting on a request. The default value is 30 seconds.
Max Request Size	Specifies the maximum number of raw bytes that can be received in a request. The default is 67108864 bytes, or 64MB. **Warning** The default file chunk size used by the IDE for file uploads is 16MB. Setting this value lower than 16MB will result in errors when uploading large files using the server manager in the IDE.
Authorized Addresses	Specifies a comma-delimited list of IP addresses that are permitted to connect to the web server. The IP addresses may include an asterisk as a wildcard character. The default value is "*", or all addresses.
Blocked Addresses	Specifies a comma-delimited list of IP addresses that are not permitted to connect to the web server. The IP addresses may include an asterisk as a wildcard character. The default value is blank (""), or no addresses.

Thread Cache Size	Specifies how many threads can be cached for the purpose of servicing connections. The default value is 128 threads.
Max Num Threads	Specifies how many threads can be created for servicing connections before the web server refuses a connection. The default value is 2048 threads.

Request Routing

The web server only supports a small number of pre-defined routes for incoming HTTP requests.



Incoming requests are routed according to the following rules:

- Incoming HEAD or GET requests whose path/resource name matches an existing file present in the configured static content directory (see the Directory Structure section above) are automatically serviced without requiring any authentication or the creation of a new session.

- If the request is not a HEAD or GET request for an existing static content file, then the request is routed based upon the path information in the request. If the initial portion of the path matches any of the pre-defined routes, then the request is routed and handled accordingly. If the request cannot be routed to any of the pre-defined routes, then a 404 Resource Not Found HTTP response code will be returned to the client performing the request.

Note
If the route is to a separate API, then that API may use additional routing rules to direct the request to the proper request handling code, whether it is internal to the web server (as with the authentication, administration, or database APIs), or whether it is implemented in a server application or native server module.

The configuration entries for the web server request routing are as follows:

Configuration Entry	Description
Keep-Alive Resource Name	Specifies the resource name for keep-alive requests. The default value is "keepalive".
Authentication Resource Name	Specifies the resource name for authentication API requests. The default value is "authentication".
Administration Resource Name	Specifies the resource name for administration API requests. The default value is "administration".
Databases Resource Name	Specifies the resource name for database API requests. The default value is "databases".
Applications Resource Name	Specifies the resource name for server application requests. The default value is "applications".
Modules Resource Name	Specifies the resource name for native server module requests. The default value is "modules".
Debugger Resource Name	Specifies the resource name for debugger API requests. The default value is "debugger".

Session Handling

As mentioned above, any HEAD or GET requests to existing static content do not require authentication and, therefore, don't require a session. However, any other requests require authentication and the creation of a session before they can be successfully serviced by the web server.

A maximum number of authentication attempts can be specified for the web server so that any authentication attempts beyond that threshold will result in the user being locked out for a specified period of time.

Sessions are handled using an HTTP-only cookie with a cryptographically-secure random session ID with a default length of 32 characters. You can use the web server Redirect to Secure setting (see the Startup section above) to ensure that sessions are always established using secure connections. By default, sessions expire after 1800 seconds, or 30 minutes, but the session expiration time can be changed, and keep-alive requests can renew a session so that the session expiration time is reset.

Note
Sessions are only created during an authentication request and are removed immediately if the authentication request fails.

The configuration entries for the web server session handling are as follows:

Configuration Entry	Description
Session ID Length	Specifies the length of the random session ID. The default value is 32 characters.
Session Expiration	Specifies, in seconds, how long a session will be kept in the web server without any activity. The default value is 1800 seconds, or 30 minutes.
Max Authentication Attempts	Specifies how many authentication attempts can be made for a user before the user is locked out. The default value is 10 attempts.
Authentication Lockout Time	Specifies how long, in seconds, a user is locked out after exceeding the maximum number of authentication attempts. The default value is 300 seconds, or 5 minutes.

Certificates

When run on Windows, the web server uses the cryptographic APIs available in the operating system for implementing secure TLS (Transport Level Security) web server connections, encryption, hashing, and random number generation. TLS is used to enable secure HTTPS connections to the web server (by default, listening on port 443). Certificates issued by a valid certificate authority are how client applications verify that they are indeed connecting to the intended host without being subjected to a man-in-the-middle attack. When an HTTPS connection is first made to the web server, a handshake occurs that provides the web server certificate along with any intermediate or root certificates that make up the chain of trust to the client. The client then verifies that these certificates are indeed valid, and disallows the connection if they are not.

On Windows, certificates are installed into a certificate store in one or more sets of certificate stores. There are separate sets of certificate stores for each user account, as well as one global set of certificate stores for the entire machine.

Certificates can be managed in several different ways:

- Programmatically using Windows API calls
- Programmatically using Windows PowerShell scripts
- Interactively using the Microsoft Management Console with the Certificates snap-in

Certificates must meet some basic requirements in order to be considered valid:

- The name of a certificate needs to match the host name that the client is trying to connect to (wildcard certificates are the one exception to this requirement). This means that the DNS entries that resolve to a specific IP address need to be correct and match the certificate name. For example, if the client is trying to connect to www.elevatesoft.com, then the DNS entries for www.elevatesoft.com should resolve to an IP address where a web server exists that will provide a valid certificate for the same www.elevatesoft.com domain.
- If you are receiving a set of certificates from a certificate authority, then you will usually receive the host certificate along with an intermediate certificate (and, possibly, a root certificate). You will need to ensure that both the host certificate and the intermediate certificate are installed in a certificate store that is accessible to the web server. Common root certificates are typically already present in the certificate stores available in Windows through a program that allows Microsoft to distribute the common root certificates. However, if this is not the case, you can still install a root certificate into a specific certificate store as an administrator, if necessary.

- Finally, installed certificates have an assigned expiration date and the host, intermediate, or root certificate cannot be expired. You can easily find out the expiration date of any installed certificate by interactively using the Microsoft Management Console or by programmatically interrogating the certificate information via PowerShell or the proper API calls.

Internal Web Server Certificate Installation

The IDE requires a localhost certificate to be installed for the current user in order to support secure HTTPS browser connections when running client browser applications in the IDE or from an external browser.

> **Warning**
> This localhost certificate should only be used for development purposes because it is not signed by a valid certificate authority.

The internal web server embedded in the IDE will not start properly without this localhost certificate being installed, and you will see an error during the IDE startup if it is not present.

The **ewbcert** utility is included to automate the process of creating the proper localhost certificate for use with the IDE. You will find it in the following installation location:

```
C:\Program Files (x86)\Elevate Web Builder 3\bin\ewbcert\win32\ewbcert.exe
```

In order to allow the localhost certificate to work correctly with all modern browsers, the ewbcert utility first creates a root certificate called "EWBRootCA", and then uses that root certificate to create/sign the localhost certificate.

Command-line usage:

```
ewbcert <Certificate Name> [-s<Certificate Store Name>] [-t<Certificate Store
      Type>] [-c|-r]

Descriptions:
-s Certificate store name, enclose in double quotes (default is "My")
-t Certificate store type: CURRENT_USER or LOCAL_MACHINE (default is
      CURRENT_USER)
-c|-r Create/remove the certificate (default is to check for certificate)
Exit codes are 0 for success, 1 for failure
```

The following command can be used to create the localhost certificate for the IDE:

```
ewbcert localhost –c
```

The following command can be used to remove the localhost certificate for the IDE:

```
ewbcert localhost -r
```

Note
If you attempt to create a localhost certificate in a LOCAL_MACHINE certificate store, you will need to do so using a command prompt running with administrator privileges or the certificate creation will fail with an "Access denied" error.

The configuration entries for the web server certificate are as follows:

Configuration Entry	Description
Certificate Name	Specifies the name of the installed certificate that will be used for secure HTTPS connections. If a certificate name is not specified, the secure server port will be disabled and a warning message will be added to the system log when the server starts. The default value is blank ("").
Certificate Store Name	Specifies the name of the certificate store where the certificate is installed. The default value is blank (""), which indicates the "My" certificate store (designated as the "Personal" store in Microsoft Management Console).
Certificate Store Type	Specifies the type of the certificate store where the certificate is installed, with 0 being a CURRENT_USER certificate store and 1 being a LOCAL_MACHINE certificate store. The default value is 0.

Cross-Origin Resource Sharing

Cross-Origin Resource Sharing is a method used by browsers and other user agents to prevent HTTP requests to web servers whose origin differs from the origin of the current URL being accessed by the browser. The origin of a URL is:

```
Protocol//Domain [:Port]
```

Browsers accomplish this by sending a pre-flight OPTIONS request to the cross-origin web server, which then responds with either an HTTP 200 status code (OK) or an error status code. By default, the web server will respond to such requests with an HTTP 400 status code (bad request). Enabling the cross-origin resource sharing in the web server will cause the web server to respond with an HTTP 200 status code and subsequently allow the cross-origin request(s).

Warning
The web server does not restrict cross-origin resource sharing by specific origins at this time. It simply allows or disallows all cross-origin requests.

The configuration entries for the cross-origin resource sharing are as follows:

Configuration Entry	Description
Enable Cross Origin Resources	Specifies that the web server will allow and handle cross-origin requests, with 1 being allow and 0 being don't allow. The default value is 0, or No.

Example Web Server Configuration File

For reference, the following is an example of a typical web server configuration file used for development with the 32-bit console version of the web server:

```
[Server]
Server Name=EWBSRVR
Server Description=Elevate Web Builder 3 Web Server
Database Directory=C:\ProgramData\Elevate Software\Elevate Web Builder 3 Web
      Server
Max Database Write Delay=5
Domain=
Default Document=
Content Directory=C:\ProgramData\Elevate Software\Elevate Web Builder 3 Web
      Server\content
Application Directory=C:\ProgramData\Elevate Software\Elevate Web Builder 3
      Web Server\applications
Module Directory=C:\ProgramData\Elevate Software\Elevate Web Builder 3 Web
      Server\modules
Log Directory=C:\ProgramData\Elevate Software\Elevate Web Builder 3 Web
      Server\logs
Logging=1
Request Logging=1
Enable Cross Origin Resources=0
Keep-Alive Resource Name=keepalive
Authentication Resource Name=authentication
Administration Resource Name=administration
Databases Resource Name=databases
Modules Resource Name=modules
Applications Resource Name=applications
Debugger Resource Name=debugger
IP Address=
Port=81
Secure Port=444
Redirect to Secure=0
Timeout=30
Max Request Size=67108864
Authorized Addresses=*
Blocked Addresses=
Thread Cache Size=128
Max Num Threads=2048
Max Authentication Attempts=10
Authentication Lockout Time=300
Session ID Length=32
Session Expiration=1800
Password Salt Length=16
Password Hash Type=0
Certificate Name=localhost
Certificate Store Name=
Certificate Store Type=0
```

2.2 Starting the Web Server

The internal web server included with the IDE can be started and stopped using the Server Manager in the IDE. Please see the Using the Server Manager topic for more information.

The external web servers included with Elevate Web Builder must be started manually, and how this is done depends upon the type of web server executable being started. The 32-bit console version of the web server can be started like a normal application, whereas both the 32-bit and 64-bit service versions of the web server must be first installed as a Windows service before they can be started using one of several methods for starting services.

> **Note**
> Installing or uninstalling the web server as a Windows service requires administrator privileges.

Installing the Web Server Service

If you wish to run the web server as a Windows service you must first install it by running the web server with the **install** command-line switch set. For example, to install the web server as a service you would specify the following command:

```
ewbsrvr.exe /install
```

To uninstall the web server as a Windows service you must run the web server with the **uninstall** command-line switch set. For example, to uninstall the web server as a service using the Run command window under Windows you would specify the following command:

```
ewbsrvr.exe /uninstall
```

Finally, by default the service will display a dialog box when the service is installed/uninstalled successfully. This is sometimes not desired during installations, and in these cases you can use the **silent** command-line switch to suppress the dialog box:

```
ewbsrvr.exe /install /silent
ewbsrvr.exe /uninstall /silent
```

Starting the 32-bit Console Web Server

Use the following command to start the 32-bit console web server from a command window:

```
ewbsrvr
```

The following is the output from the command:

```
Elevate Web Builder Command-Line Web Server
```

```
Server: EWBSRVR [Elevate Web Builder 3 Web Server]
Listening on port 81 and secure port 444

Database: C:\ProgramData\Elevate Software\Elevate Web Builder 3 Web Server
Content: C:\ProgramData\Elevate Software\Elevate Web Builder 3 Web
      Server\content
Applications: C:\ProgramData\Elevate Software\Elevate Web Builder 3 Web
      Server\applications
Modules: C:\ProgramData\Elevate Software\Elevate Web Builder 3 Web
      Server\modules
Logs: C:\ProgramData\Elevate Software\Elevate Web Builder 3 Web Server\logs
Event logging enabled
Request logging enabled

Press <Enter> to exit
```

Press the **Enter** key to stop the 32-bit console web server and close the command window.

Starting the 32-bit or 64-bit Web Server Service

To start the 32-bit or 64-bit web server service, use the **net start** command from a command window:

```
net start ewbsrvr
```

You can also interactively start the web server service using the Microsoft Management Console and the Services snap-in.

Stopping the 32-bit or 64-bit Web Server Service

To stop the 32-bit or 64-bit web server service, use the **net stop** command from a command window:

```
net stop ewbsrvr
```

You can also interactively stop the web server service using the Microsoft Management Console and the Services snap-in.

Note
In order to start and stop the web server service, the web server must have already been installed as a Windows service using the **install** command-line switch (see above). Also, you must have administrator privileges in order to start and stop the web server service.

2.3 Multiple Web Server Instances

Multiple instances of the web server can be run on the same physical machine by using named server instances. Named server instances are simply instances of the web server that were executed using two special command-line switches:

```
ewbsrvr.exe /name=<Server Name> /desc=<Server Description>
```

Named server instances use the passed name and description to provide the name of the web server instance, as well as the description. The **name** parameter is also used to determine which section of the configuration .ini file is used for configuring the web server instance. Instead of just the normal "Server" section being used in the configuration .ini file, the section is named using the provided server name. For example, if the named server instance is called "MyServer", then the section in the configuration .ini file where the configuration is stored will be the following:

```
[Server_MyServer]
```

The **description** parameter, if also specified, is immediately written to **Server Description** configuration entry key under the named server instance section of the configuration .ini file. All other configuration options must be modified manually.

Named Server Instances and Services

In order to use a named server instance as a Windows service, the **name** parameter must be specified during the installation of the service. For example, if the named server instance is called "MyServer", then the service installation would be accomplished using the following from the command-line:

```
ewbsrvr.exe /install /name=MyServer /desc="My Server"
```

When you want to start the named server instance as a service, you would simply just use the following from the command-line:

```
net start MyServer
```

The following example shows how you would install two named server instances as services, and then start them:

```
ewbsrvr.exe /install /name=MyFirstServer /desc="My First Server"

ewbsrvr.exe /install /name=MySecondServer /desc="My Second Server"

net start MyFirstServer

net start MySecondServer
```

Note
There is no limit to the number of server instances that can be started, but each instance must use a different server name and set of insecure and secure ports.

2.4 Web Server Request Handling

The web server is capable of servicing standard HTTP 1.1 requests, both for static content and dynamic content provided via server applications/native server modules. A typical HTTP 1.1 request looks like this:

```
GET /musiccollection.html HTTP/1.1
Accept: text/html
Accept-Encoding: gzip, deflate
Accept-Language: en-us
Cache-Control: max-age=0
Connection: keep-alive
Host: localhost
If-Modified-Since: Thu, 16 Aug 2012 18:35:21 GMT
User-Agent: Mozilla/5.0
```

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

Method Definitions

The web server supports the GET, HEAD, POST, PUT, PATCH, and DELETE HTTP methods in web server requests:

HTTP Method	Description
GET	Used to retrieve a resource from the web server.
HEAD	Used to retrieve meta-information about a resource on the web server, and is identical to a GET request but does not include the resource in the response.
POST	Used to submit content to a resource on the web server.
PUT	Used to replace a resource on the web server.
PATCH	Used to partially update a resource on the web server.
DELETE	Used to delete a resource from the web server.

URL parameters are specified as part of the URL in the following manner:

```
<BaseURL>?<Params>

<Params> = <Param> [&<Param>]

<Param> = <Key>=<Value>
```

For example, the following URL contains two parameters for X and Y axis values:

```
https://www.elevatesoft.com/applications/compute?x=100&y=200
```

After the initial request line is one carriage return/line feed pair (0x0D and 0x0A), followed by the request headers. All request headers use a format of:

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

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

Header Field Definitions

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

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

For example, suppose that you want to submit the following text content to a resource on the web server using a POST request:

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

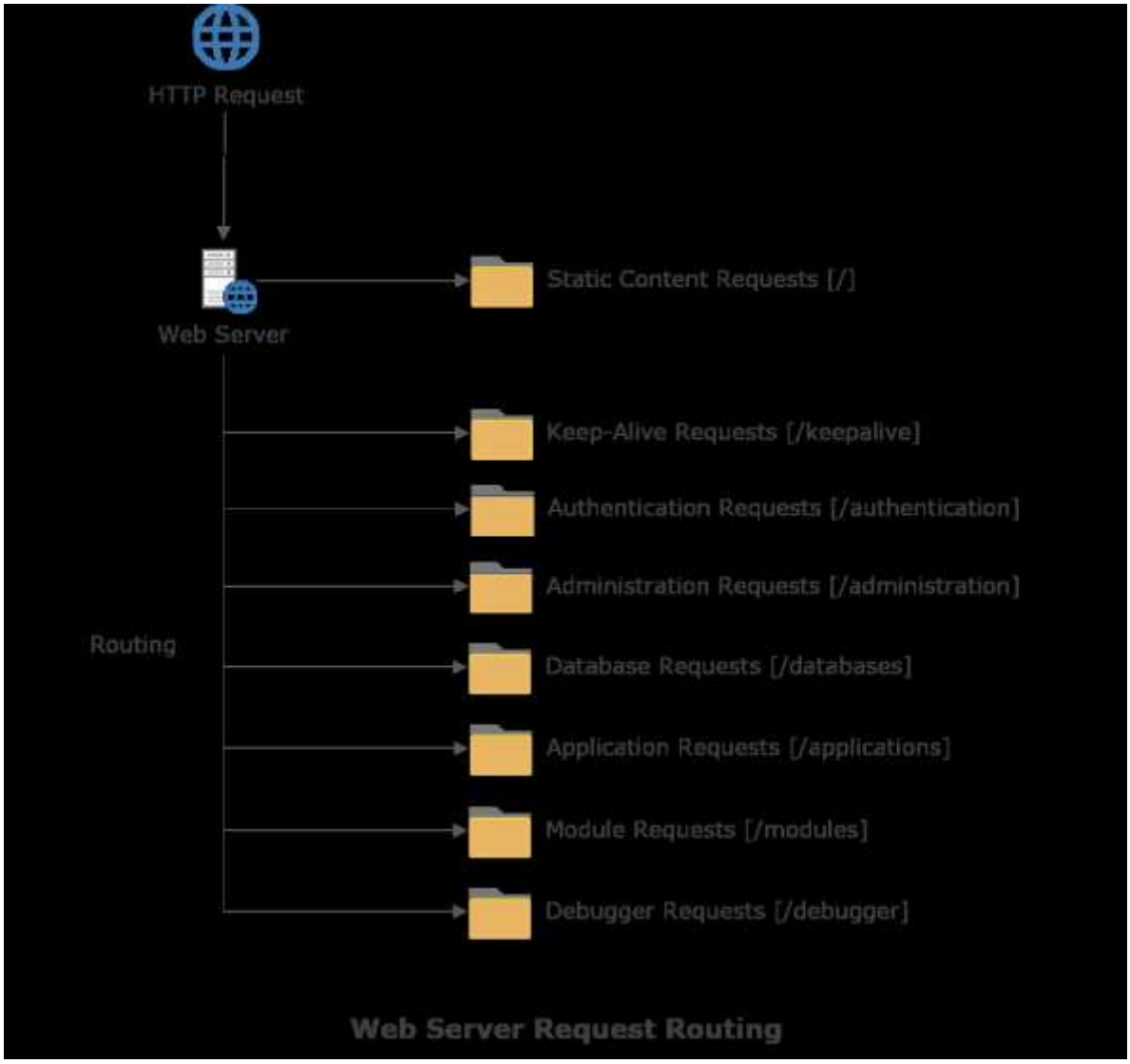
The length of the text is 43 single-byte characters, so the POST request would look like this:

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

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

Note
You do not have to format web server requests like this in order to execute such requests in either client or server applications. However, it is important that you understand how web server requests are formatted in order to properly add custom headers or content to web server requests, as well as to properly read and parse response content returned from the web server.

Web Server Request Routing

Incoming web server requests include a URL that indicates the resource for the request. This resource can include path information that, along with the configuration of a small number of pre-defined routes in the web server, determines how the request is routed within the web server.



The web server requests are routed according to the following rules:

- Incoming HEAD or GET requests whose path/resource name matches an existing file present in the configured static content directory are automatically serviced without requiring any authentication or the creation of a new session.

- If the request is not a HEAD or GET request for an existing static content file, then the request is routed based upon the path information in the request. If the initial portion of the path matches any of the pre-defined routes, then the request is routed and handled accordingly. If the request cannot be routed to any of the pre-defined routes, then a 404 Resource Not Found HTTP response code will be returned to the client performing the request.

> **Note**
> If the route is to a separate API, then that API may use additional routing rules to direct the request to the proper request handling code, whether it is internal to the web server (as with the authentication, administration, or database APIs), or whether it is implemented in a server application or native server module.

Please see the Configuring the Web Server topic for more information on configuring the resource names for the web server.

Web Server Responses

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

```
HTTP/1.1 200 OK
Date: Thu, 17 Aug 2012 01:52:46 GMT
Server: Elevate Web Builder 3 Web Server
Connection: keep-alive
Cache-Control: no-cache
Content-Type: text/plain; charset=utf-8
Content-Length: 43

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

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

Status Code and Reason Phrase

Web Server Content Handling

Both web server requests and their responses can include content in many different formats and encodings. Several different HTTP headers are responsible for specifying information about the included content.

Content-Type

The **Content-Type** header specifies the MIME type of content being included with either the request or response. Some common MIME types are:

MIME Type	Description
text/plain	Plain text
text/html	HTML
text/xml	XML
application/xml	XML
application/json	JSON
application/javascript	JavaScript
application/octet-stream	Binary data
application/x-www-form-urlencoded	HTML form values
multipart/mixed	Multiple content types combined
multipart/form-data	HTML form values and file upload data combined
image/bmp	Bitmap raster image
image/gif	GIF raster image
image/jpeg	JPEG raster image
image/png	PNG raster image

image/tiff	TIFF raster image
image/svg+xml	SVG vector image
application/x-zip-compressed	Zip archive
application/x-gzip-compressed	GZip archive
application/pdf	PDF document
audio/aac	AAC audio
audio/mp4	MP4 audio
audio/mpeg	MP3 audio
audio/ogg	OGG audio
audio/wav	WAV audio
video/mpeg	MPEG video
video/mp4	MP4 video
video/ogg	OGG video
video/quicktime	QuickTime video
video/x-ms-wmv	Windows Media video
video/x-flv	FLV video
video/webm	WEBM video
application/x-font-truetype	TrueType font
application/x-font-opentype	OpenType font
application/font-woff	WOFF web font
application/font-woff2	WOFF2 web font

The **Content-Type** header can also include character set information that further indicates how the characters included in any textual content are encoded. For example, plain text encoded using the UTF-8 character set would have the following header:

```
Content-Type: text/plain; charset=utf-8
```

By default, the web server (along with the IDE) always use the UTF-8 character set for encoding any textual content, and assumes the UTF-8 character encoding for any text that doesn't explicitly include a character set encoding.

Content-Length

The **Content-Length** header specifies the size of the included content in bytes.

When executing web server requests from a client browser application, the **Content-Length** header is automatically set by the browser. Please see the Server Request Architecture for more information on how content is handled when executing web server requests in this context.

When returning a response in a server application or native server module, the **Content-Length** header is automatically set by the web server.

Content-Encoding

The **Content-Encoding** header specifies any additional encoding of the included content. The web server supports

the following content encodings:

Encoding	Description
gzip	Compressed using the GZip format
deflate	Compressed using the ZLib format

Warning
The deflate encoding is problematic when used with the older Internet Explorer browsers and other older Microsoft server products. The newer specifications for the deflate encoding indicate that the encoding should use the ZLib format, whereas the older specifications used by Internet Explorer and other products indicate that the encoding should use a raw deflate algorithm without the ZLib headers/footers (including checksums). For this reason, the web server prioritizes using the gzip encoding over the deflate encoding.

The **Accept-Encoding** header can be included with any request by a user agent to indicate which encodings are supported by the user agent. The web server uses this information to determine which encoding, if any, to use for the content included with a response. By default, the web server will use the gzip or deflate encodings for any response content if the following two conditions are met:

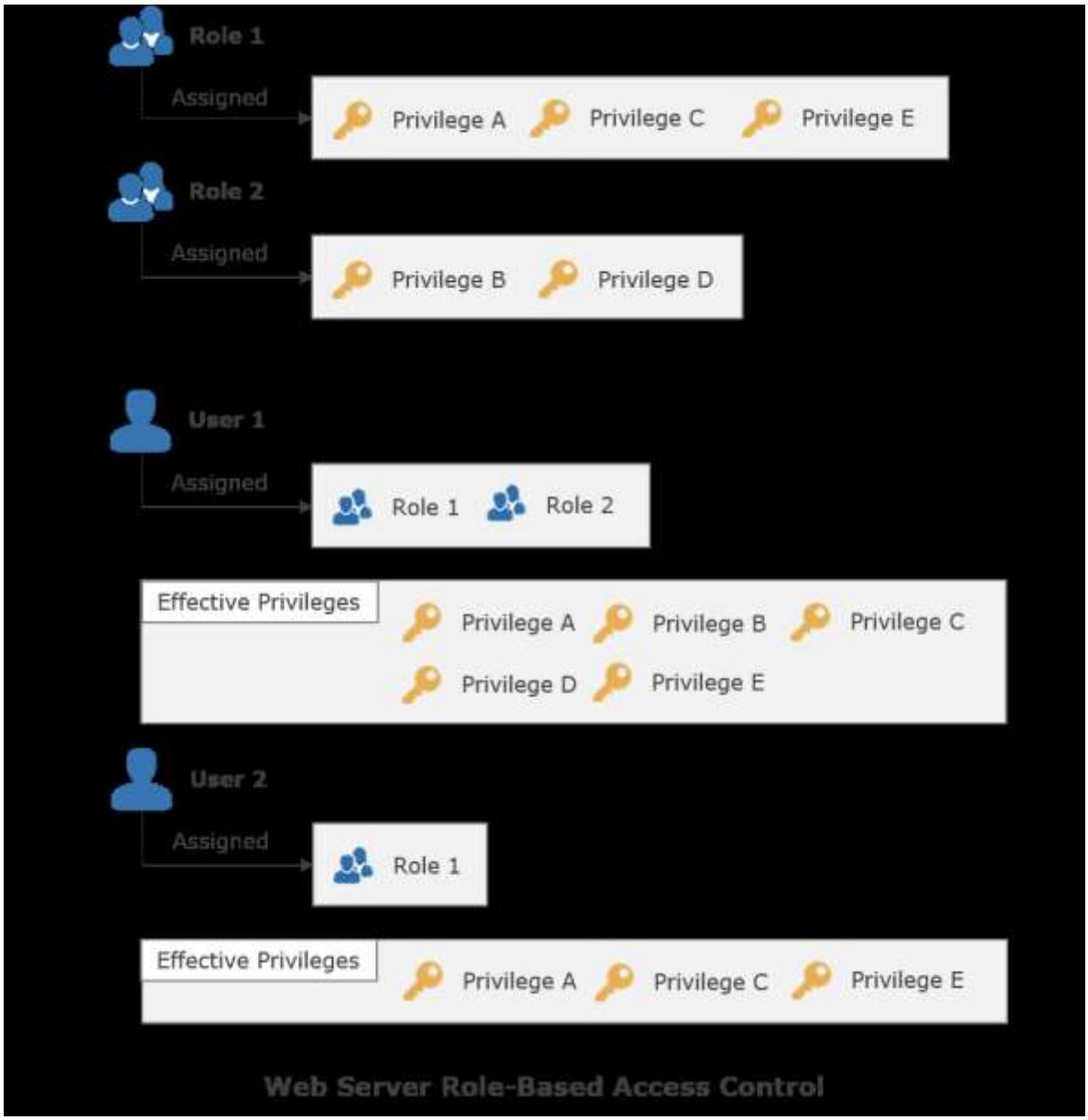
- The response content is textual, meaning that the response **Content-Type** header is set to one of the following values:

MIME Type	Description
text/plain	Plain text
text/html	HTML
text/xml	XML
application/xml	XML
application/json	JSON
application/javascript	JavaScript
application/pdf	PDF document

- The user agent has indicated that it will accept either the gzip or deflate encodings via an **Accept-Encoding** header.

2.5 Web Server Security

The web server uses a role-based access control (RBAC) architecture:



There are three main functional components to this architecture:

- Privileges
- Roles
- Users

Privileges are assigned to roles, which are then assigned to users. Both roles and users can be designated as active or inactive. The net effect is that each user has a set of effective privileges that are the union of all privileges granted to all of the active roles that are granted to the user. Once a user authenticates against the web server, this set of effective privileges is assigned to the web server session that is created during authentication. Please see the Web Server Authentication topic for more information on how sessions are created during authentication.

Privileges are used to secure access to various server objects like databases, dataset commands, server applications, and native server modules. Each of these server objects can be assigned a privilege that determines whether the object is accessible to the user.

> **Warning**
> If a server object is not assigned a privilege, then it is assumed to be globally accessible to any user. Also, server applications have access to all databases and datasets, regardless of the authenticated user that is making the request to the server application. This makes it very important to monitor and audit who is designated as being able to install/uninstall server applications in the web server.

Privileges can also provide access control within server applications. Server applications can check to see if the user executing a server application request has been granted a given role or privilege using one of the following methods of the incoming web server request instance:

HasRole
HasPrivilege

Please see the Server Applications topic for more information on how to access the properties and methods of incoming web server requests in server applications.

The web server is automatically configured with a set of pre-defined privileges, roles, and users that are primarily used for securing administrative access to the web server. They provide fine-grained control over what administrative functionality is granted to any given role.

Privileges

The following is the list of pre-defined privileges in the web server, separated by functional group:

Privilege Management

Privilege	Description
AddPrivilege	Allows the user to add new privileges.
UpdatePrivilege	Allows the user to update existing privileges.
RemovePrivilege	Allows the user to remove existing privileges.
GetPrivileges	Allows the user to enumerate all privileges.

Role Management

Privilege	Description
AddRole	Allows the user to add new roles.
UpdateRole	Allows the user to update existing roles.
RemoveRole	Allows the user to remove existing roles.
UpdateRoleActiveStatus	Allows the user to update the active status of existing roles.
UpdateRolePrivileges	Allows the user to update the assigned privileges of existing roles.
GetRolePrivileges	Allows the user to enumerate the assigned privileges of existing roles.
GetRoles	Allows the user to enumerate all roles.

User Management

Privilege	Description
AddUser	Allows the user to add new users.
UpdateUser	Allows the user to update existing users.
RemoveUser	Allows the user to remove existing users.
UpdateUserActiveStatus	Allows the user to update the active status of existing users.
UpdateUserLockStatus	Allows the user to update the lock status of existing users.
UpdateUserPassword	Allows the user to update the password of existing users.
UpdateUserRoles	Allows the user to update the assigned roles of existing users.
GetUserRoles	Allows the user to enumerate the assigned roles of existing users.
GetUsers	Allows the user to enumerate all users.

Database Management

Privilege	Description
TestDatabase	Allows the user to test database connections.
AddDatabase	Allows the user to add new databases.
UpdateDatabase	Allows the user to update existing databases.
RenameDatabase	Allows the user to rename existing databases.
RemoveDatabase	Allows the user to remove existing databases.
GetDatabasePrivileges	Allows the user to get the assigned privileges for an existing database.
SetDatabasePrivileges	Allows the user to set the assigned privileges for an existing database.
GetDatabaseTables	Allows the user to enumerate all base tables in an existing database.
GetDatabases	Allows the user to enumerate all databases.
AddDataSet	Allows the user to add new datasets to an existing database.
UpdateDataSet	Allows the user to update existing datasets in an existing database.
RenameDataSet	Allows the user to rename existing datasets in an existing database.
RemoveDataSet	Allows the user to remove existing datasets in an existing database.
AddDataSetCommand	Allows the user to add new commands to an existing dataset.
UpdateDataSetCommand	Allows the user to update existing commands in an existing dataset.
RenameDataSetCommand	Allows the user to rename existing commands in an existing dataset.
RemoveDataSetCommand	Allows the user to remove existing commands in an existing dataset.
GetDataSetCommandPrivileges	Allows the user to get the assigned privileges for an existing command.

SetDataSetCommandPrivileges	Allows the user to set the assigned privileges for an existing command.
GenerateDataSetCommands	Allows the user to generate the commands for an existing dataset.

Native Server Module Management

Privilege	Description
InstallModule	Allows the user to install a native server module.
RenameModule	Allows the user to rename an existing installed native server module.
UninstallModule	Allows the user to uninstall an existing installed native server module.
GetModulePrivileges	Allows the user to get the assigned privileges for an existing installed native server module.
SetModulePrivileges	Allows the user to set the assigned privileges for an existing installed native server module.
GetModules	Allows the user to enumerate all installed native server modules.

Server Application Management

Privilege	Description
InstallApplication	Allows the user to install a server application.
RenameApplication	Allows the user to renaming an existing installed server application.
UninstallApplication	Allows the user to uninstall an existing installed server application.
GetApplicationPrivileges	Allows the user to get the assigned privileges for an existing installed server application.
SetApplicationPrivileges	Allows the user to set the assigned privileges for an existing installed server application.
GetApplications	Allows the user to enumerate all installed server applications.
DebugApplication	Allows the user to debug an existing installed server application.

Server Status

Privilege	Description
GetServerStatus	Allows the user to retrieve various server statistics.

Deployment and Folder/File Management

Privilege	Description

UploadFiles	Allows the user to upload files to the web server.
DownloadFiles	Allows the user to download files from the web server as a .zip file.
RenameFile	Allows the user to rename existing files on the web server.
RemoveFile	Allows the user to remove existing files on the web server.
CreateFolder	Allows the user to create new folders on the web server.
RenameFolder	Allows the user to rename existing folders on the web server.
RemoveFolder	Allows the user to remove existing folders on the web server.
GetFiles	Allows the user to enumerate all files in an existing folder.

Roles

The following is the list of pre-defined roles in the web server and their assigned privileges:

Role	Default Assigned Privileges
Administrators	All of the pre-defined privileges
Public	None

Users

The following is the list of pre-defined users in the web server and their assigned roles:

User	Default Assigned Roles
Administrator	Administrators
Anonymous	Public

Default Authentication Information

The pre-defined **Administrator** user uses the following password (case-sensitive):

```
EWBDefault
```

Warning
You should not deploy instances of the web server into production without first ensuring that such instances have been properly configured so that the default Administrator password has been changed. For both the 32-bit and 64-bit web server services, but not the console version of the web server included for development purposes, the web server will log a warning into the system log if the default password is still set for the pre-defined Administrator user. Please see the Web Server Logging topic for more information.

The pre-defined **Anonymous** user does not have a password.

2.6 Web Server Logging

The web server implements two different types of logging:

Log Type	Description
HTTP Request	Logs basic information about all HTTP requests and responses.
System	Logs extensive information about all authentication and administration requests, the execution of any dataset commands that have been configured to have their execution logged, and any errors that occur during the execution of any server applications or native server modules.

Please see the Configuring the Web Server topic for more information on how to enable or disable either type of logging, as well as how to configure the directory where the log files are stored.

HTTP Request Logging

By default, the HTTP request logging is enabled in the web server and outputs a daily log of all HTTP requests in the CSV (comma-separated value) format into the following log directory:

```
<Log Directory>\Request
```

The file name is formatted using a 4-digit year, a 2-digit month, and a 2-digit day:

```
<Year><Month><Day>.csv
```

Each HTTP request occupies a single line in the .csv log file followed by a CR-LF line terminator. The format of each line in the log is as follows:

Value	Description

Date-Time	The date/time of the request as a string in HTTP header date/time format (RFC2616).
Client IP Address	The IP address of the user agent making the request as a string.
Protocol	The protocol of the request (HTTP or HTTPS) as a string.
Host	The Host request header for the request as a string.
Version	The HTTP version of the request as a string (should always be "HTTP/1.1").
Method	The HTTP method as a string.
URL	The request URL as a string.
User Agent	The User-Agent request header for the request as a string.
Referer	The Referer request header for the request as a string.
Content Type	The Content-Type request header for the request as a string.
Content Length	The Content-Length request header for the request as an integer.
Execution Time	The execution time, in milliseconds, of the request as an integer. **Note** The execution time includes all processing up to, but not including, the actual sending of the response to the request.
Response Status	The HTTP status code of the response as an integer.
Response Status Message	The HTTP status message of the response as a string.
Response Content Type	The Content-Type response header as a string.
Response Content Length	The Content-Length response header as an integer.

System Logging

By default, the system logging is enabled in the web server and outputs a daily log of all web server events in the JSON format into the following log directory:

```
<Log Directory>\System
```

The file name is formatted using a 4-digit year, a 2-digit month, and a 2-digit day:

```
<Year><Month><Day>.elog
```

Each logged event occupies a single line in the .elog log file followed by a CR-LF line terminator. Each line in the log is formatted as a standalone JSON object with the following properties:

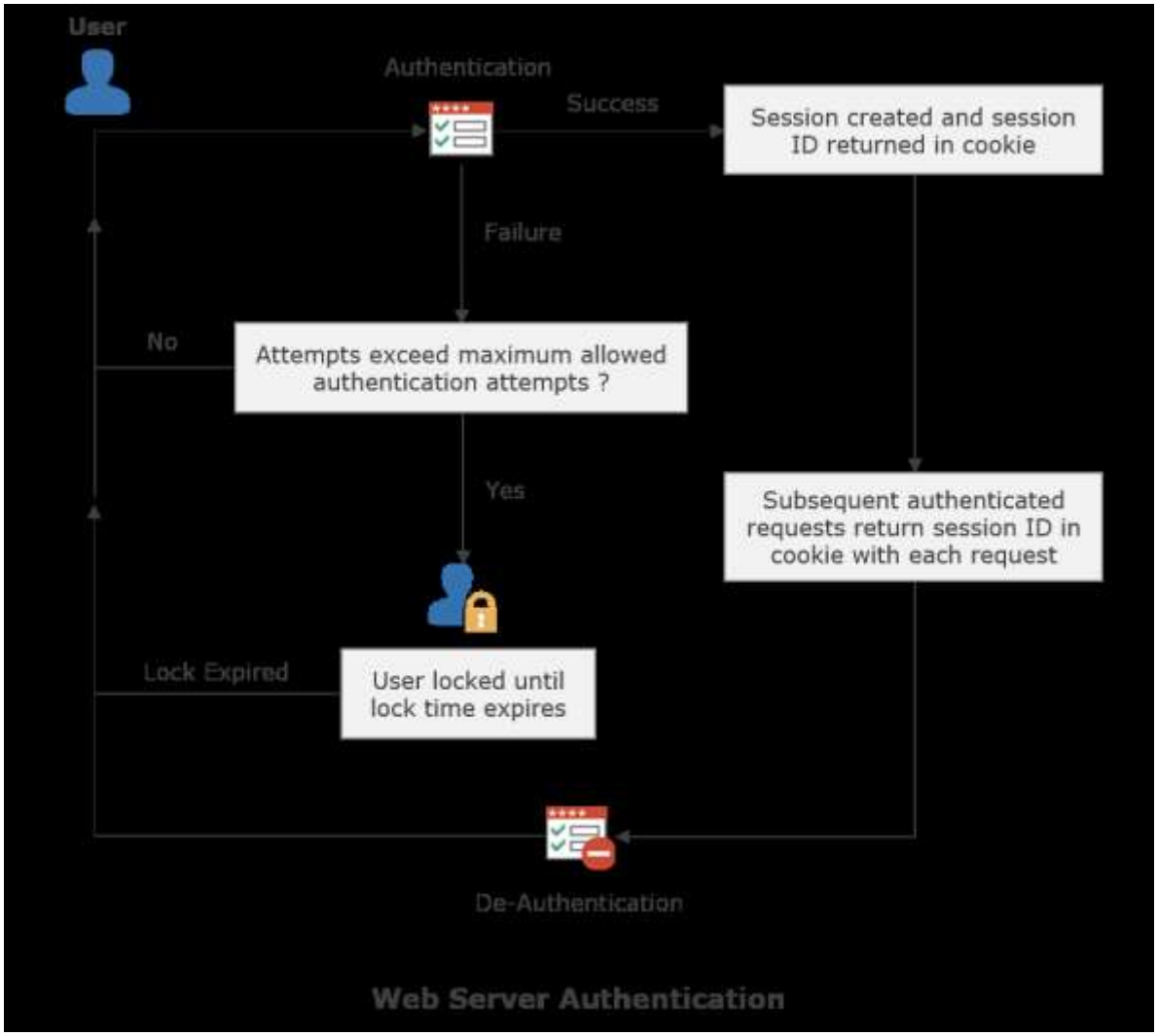
Property	Description

Time	The date/time of the web server event in ISO 8601 format as a string.
Category	The category of the web server event as a string: "Info", "Warning" or "Error".
ID	The ID of the web server event as a string.
Details	The details of the web server event as a nested JSON object. The details logged are specific to each web server event.

Warning
The format of the system log files is non-standard for JSON files, which is why the web server uses the .elog extension instead of the .json extension for the log files. The system log files are formatted this way in order to avoid an outer JSON object in the file that would make reading of a log file problematic while that same log file was also concurrently being written to.

2.7 Web Server Authentication

The web server ensures that unauthorized access does not occur by requiring that any access to non-static content require authentication with a valid user name and password. Please see the Web Server Security topic for more information on the default users available in the web server.



The authentication process allows any client application to send an authentication request with a user name and password to the web server. If the user name and password are correct and the user is not inactive or locked out due to previously exceeding the maximum number of invalid authentication attempts, then the authentication will succeed and a new session will be created. The session ID of this new session will be sent back to the client application in a special HTTP-only cookie called **EWBSessionID**.

Warning
This cookie is not accessible to JavaScript code in browsers, which makes it safe from session hijacking in a browser environment. However, if the connection to the web server is an insecure HTTP connection, then this cookie can be intercepted as plain text during transmission between the web server and the client application. Please see the Configuring the Web Server topic for more information on how to configure the web server to automatically redirect all requests to the insecure HTTP port to the secure HTTPS port.

Note
If the current session was previously manually deauthenticated using a deauthenticate request, then any subsequent authentication request will not create a new session provided the previous session cookie was sent by the client application along with the subsequent authentication request. The only exception to this is if the session has expired since the manual deauthentication request.

The session created during authentication will remain active until there have been no requests for the session during the amount of time specified by the configured session expiration time on the web server. Please see the Configuring the Web Server for more information on setting the session expiration time.

Every time a request is made using a given session, the session expiration time is reset. In order to manually reset the session expiration time and keep a session active during periods of inactivity, the client application can execute an HTTP POST request to the following keep-alive URL:

```
<Origin>/keepalive

<Origin> = <Protocol>://<Domain>[:<Port>]
```

Note
The "keepalive" resource name is the default resource name for keep-alive requests. The keep-alive resource name is configurable in the web server, so please make sure that the resource name that you are using matches the proper resource name on the target web server.

2.8 Web Server Authentication API

The authentication API requests use a remote procedure call style that is structured as follows:

```
<Origin>/authentication?method=<Method>

<Origin> = <Protocol>://<Domain>[:<Port>]

<Method> = Authentication method name
```

Note
The "authentication" resource name is the default resource name for authentication requests. The authentication resource name is configurable in the web server. Please make sure that the resource name that you are using matches the proper resource name on the target web server.

Any content included with authentication API requests, or returned as a response to the request, should/will be formatted as JSON content. The date/time format used in the JSON content is equivalent to a raw JavaScript Date value: an integer value representing the number of milliseconds since 1 January 1970 UTC.

Note
Any forward slashes present in any JSON content returned as a response by the authentication API will be escaped so that the JSON content can be included in HTML without issues.

Authenticate Request

The authenticate request authenticates the user and password provided in the included JSON content. If a session is already active and the user is authenticated, then the user will be deauthenticated before being authenticated. If the authentication fails for any reason, the request will result in a 500 Internal Error HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/authentication?method=authenticate
```

Example Request Content:

```
{
   "User": "Administrator",
   "Password": "EWBDefault"
}
```

HTTP Response: 200 on success or 500 on error

Deauthenticate Request

If a session is already active and the user is authenticated, then the user will be deauthenticated. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

Example Request:

```
https://localhost/authentication?method=deauthenticate
```

Example Request Content: None

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Get Effective Access Request

The get effective access request retrieves the currently assigned roles and privileges for an authenticated user. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: GET

HTTP Response Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/authentication?method=geteffectiveaccess
```

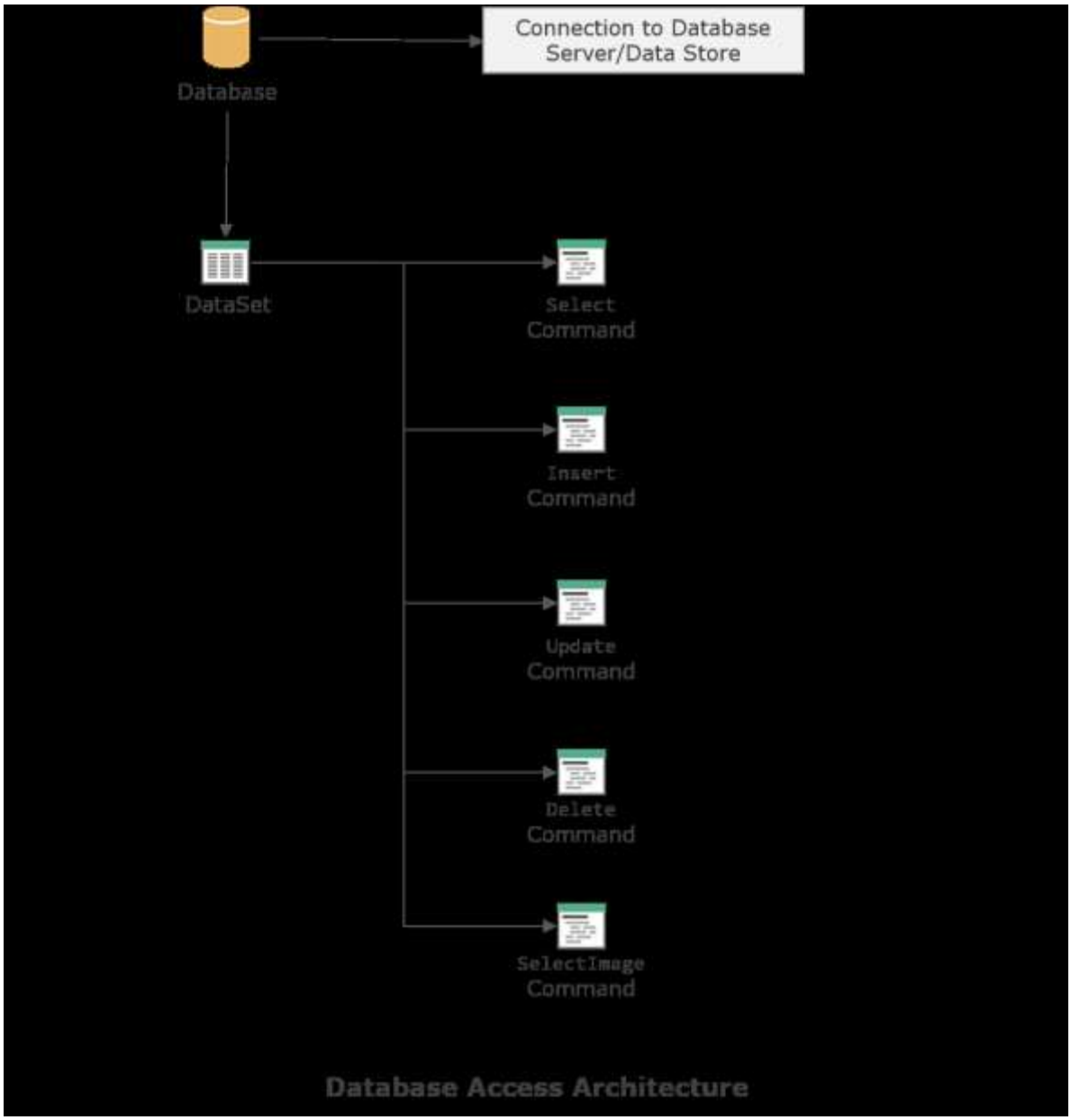
Example Response Content:

```
{
   "Roles": ["Developers", "Public"],
   "Privileges": ["UploadFiles", "DownloadFiles", "GetFiles",
      "DebugApplication"]
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

2.9 Web Server Database Access

The web server includes database access functionality that can be used to allow authenticated users to read and write the data stored on one or more private database servers/stores in a safe and secure manner.



Databases

A database is a virtual object that represents a database connection to a target database. Each database is defined with a database engine type along with a set of engine-specific connection properties that specify how to succesfully connect to the target database.

The following types of database connections are supported:

- ElevateDB
- DBISAM

- ADO/ODBC

Each database can be assigned an access privilege that determines if a database is accessible for a given user. Please see the Web Server Security topic for more information on roles and privileges.

DataSets

Within each database are datasets, which are virtual objects that represent a set of rows. DataSets are purely virtual and are only containers for a set of dataset commands. Each dataset can be configured so that all date/time column values are localized when read/written from/to the underlying database table(s).

DataSet Commands

Within each dataset are dataset commands, which are containers for engine-specific SQL statements/calls that determine which commands can be executed for the dataset. These SQL statements/calls use named input parameters (:ParamName) to control which rows are selected, updated, or deleted, and named input-output or output parameters (:ParamName) to retrieve autoinc/identity/generated column values after rows are inserted or updated during a transaction. There are two types of dataset commands:

Command Type	Description
Dataset	These commands execute SQL statements that perform row operations such as SELECT, INSERT, UPDATE, and DELETE DML statements.
BLOB Column	These commands execute SQL statements that perform BLOB column operations such as SELECT and UPDATE DML statements, as well as retrieving the MIME content type of BLOB columns, and are named according to the BLOB column that is being operated on. This architecture optimizes BLOB column access performance and allows for the selective updating of BLOB columns.

There are four supported dataset commands:

Command Name	Description
Select	Contains an SQL statement that returns a result set. If the user cannot execute the Select command, then the dataset is effectively "invisible" to the user unless the dataset is being accessed in a server application. **Note** In addition to this command, you will also need to add column-level commands for returning the data from any BLOB columns that are present in the result set, as specified below
Insert	Contains an SQL statement that inserts a new row into the dataset. If the user cannot execute the Insert command, then the user cannot insert any rows into the dataset unless the dataset is being updated in a server application. **Note** You should not include BLOB columns in the SQL statement. Instead, specify any BLOB columns as individual column updates as specified below.

Update	Contains an SQL statement that updates an existing row in the dataset. If the user cannot execute the Update command, then the user cannot update any rows in the dataset unless the rows are being updated in a server application. **Note** You should not include BLOB columns in the SQL statement. Instead, specify any BLOB columns as individual column updates as specified below.
Delete	Contains an SQL statement that deletes an existing row in the dataset. If the user cannot execute the Delete command, then the user cannot delete any rows in the dataset unless the rows are being deleted in a server application.

There are two supported BLOB column commands, with a variation for the Update command that is used to update the content type for the BLOB column in a given row:

Command Name	Description
Select<Column Name>	Contains an SQL statement that returns a result set containing the specified BLOB column and, optionally, the MIME content type column for the BLOB column data, in a single row. If the user cannot execute the column command, then the user cannot read the BLOB column unless the BLOB column is being read in a server application. The MIME content type column is used by the web server to assign the **Content-Type** response header for any access to the BLOB column. The name of the MIME content type column should be set to <Column Name>_ContentType in order for the MIME content type column to be used correctly by the database access in the web server. **Note** The BLOB MIME content type column is optional. However, if it does not exist, then the user agent or server application retrieving the data will need to be able to determine the format of the data through other means.
Update<Column Name>	Contains an SQL statement that updates the specified BLOB column in an existing row in the dataset. **Note** This type of command only applies to server applications. It is currently not possible to update BLOB columns using client applications.
Update<Column Name>_ContentType	Contains an SQL statement that updates the MIME content type for the specified BLOB column in an existing row. This content type is used by the web server to assign the **Content-Type** response header for any access to the BLOB column.

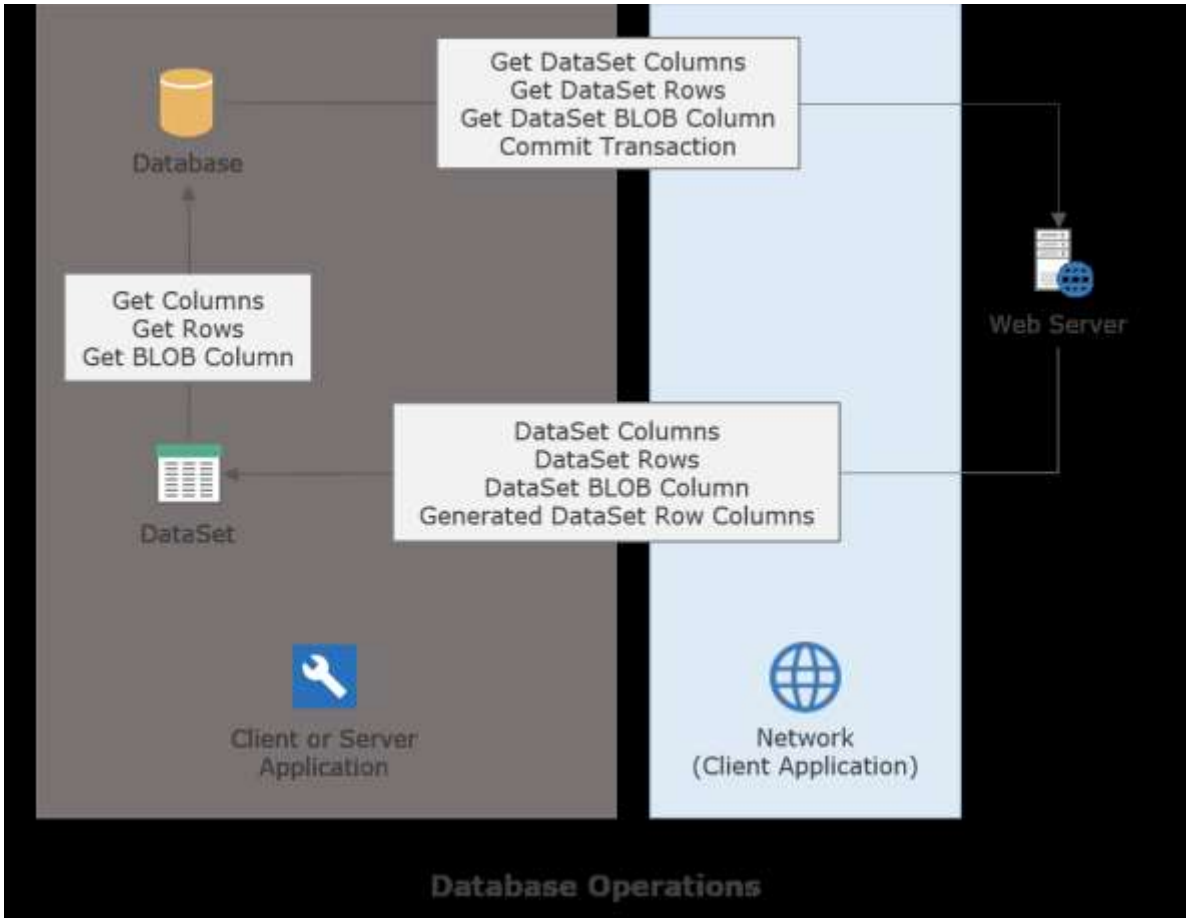
	Note This type of command is optional. If you do not wish to allow an update to the MIME content type for a BLOB column, then you do not need this type of command for the BLOB column.

Note
The dataset command names must match (case-insensitive) the above command names in order for them to be executed correctly for the dataset.

The dataset commands can be generated for a given dataset using the administration API. Please see the Web Server Administration API - Databases topic for more information.

Each dataset command can be assigned an execution privilege that determines if a dataset is accessible for a given user, or whether the user can execute a given dataset command. Please see the Web Server Security topic for more information on roles and privileges.

Database Operations



The database operations are universal in the web server, meaning that they function very similarly for both client and server applications. The primary differences between the client and server application database operations are:

- Database operations in client applications require an authenticated session in order to complete successfully. Database operations in server applications are already operating under an authenticated session. However, database access in server applications is unrestricted, and server applications do not use the effective privileges of the authenticated session in order to determine if any given database is accessible. Server applications behave as if they are executing as a "super user" with respect to database access.

> **Note**
> It is always recommended that you implement additional privileges for accessing functionality in server applications and use them to restrict access to any personal information or other types of sensitive data. It is important to remember that users can authenticate against the web server as the built-in Anonymous user. Please see the Web Server Security topic for more information.

- BLOB columns can only be updated using server applications. Client applications can only read BLOB data and display it using a properly-constructed BLOB column database access API URL. These URLs are constructed automatically in client applications by the database access components. Please see the Web Server Database Access API topic for more information on how the database access API URLs are structured.

There are five types of database operations used in the web server:

Operation	Description
Get DataSet Columns	This operation executes the **Select** dataset command (without any input parameters) and returns the columns for the dataset.
Get DataSet Parameters	This operation executes the **Select** dataset command (without any input parameters) and returns the parameters for the command.
Get DataSet Rows	This operation executes the **Select** dataset command and returns the rows for the dataset.
Get DataSet BLOB Column	This operation executes the **Select<Column Name>** dataset command and returns the data in the specified BLOB column along with an optional <Column Name>_ContentType column that specifies the MIME content type for the BLOB column data.
Commit Transaction	This operation commits a transaction that was previously started automatically or manually by the client or server application. Each transaction contains a series of row inserts, updates, and deletes that are executed in the order in which they occurred using the **Insert**, **Update**, **Update<Column Name>**, and **Delete** commands. For server applications, a transaction commit will update any BLOB columns that were modified after each insert or update. **Note** Each transaction commit is bracketed with an actual database transaction for the underlying database connection, making all transactions atomic.

Transactions can be nested in both client and server applications, and can be started automatically or manually. If automatic transactions are enabled (the default), then any insert, update, or delete operation in the dataset will cause a transaction to be started. After the insert, update, or delete operation completes successfully, the automatic transaction is then committed. Please see the Transactions topic for more information.

2.10 Web Server Database Access API

The database API requests use a REST-ful style that is structured as follows:

```
<Origin>/databases/<Database>[/<DataSet>[/Data]]?<Parameters>

<Origin> = <Protocol>://<Domain>[:<Port>]

<Database> = Database name

<DataSet> = DataSet name

<Parameters> = URL parameters in key=value form and an
               ampersand (&) between parameters
```

Note
The "databases" resource name is the default resource name for database requests. The database resource name is configurable in the web server. Please make sure that the resource name that you are using matches the proper resource name on the target web server.

Any content included with database API requests, or returned as a response to the request, should/will be formatted as JSON content. The date/time format used in the JSON content is equivalent to a raw JavaScript Date value: an integer value representing the number of milliseconds since 1 January 1970 UTC.

Note
Any forward slashes present in any JSON content returned as a response by the database API will be escaped so that the JSON content can be included in HTML without issues.

Get DataSet Columns Request

The get dataset columns request enumerates all of the columns in the dataset and database specified in the URL. The column information returned includes the name, type, length (if applicable), and scale (if applicable). The various supported column types are:

Column Type	Description

0	Unknown type - will cause an error when the columns are loaded
1	String - requires a column length for fixed-length columns, null for variable-length columns
2	Boolean
3	Integer
4	Float - can have a column scale specified
5	Date
6	Time
7	Date/Time
8	BLOB
9	CLOB (non-browser clients only)

For browser client applications, BLOB columns in datasets are defined like variable-length String columns with a null length, and contain a URL that is used to dynamically load the BLOB column from the client application. However, CLOB columns (Character Large Object) in datasets are defined as actual String columns with a null length. For an example of a URL for a BLOB column request, please see the Get DataSet BLOB Column request below.

For non-browser client applications, BLOB columns in datasets are also defined like variable-length String columns with a null length, and contain a URL that is used to dynamically load the BLOB column from the client application. However, CLOB columns (Character Large Object) in datasets are defined with a distinct CLOB type, and are handled like a String column with a null length.

> **Note**
> BLOB columns cannot be updated from browser client applications, whereas CLOB columns can be updated.

Also, an optional additional String column can be defined for a BLOB column that indicates the MIME type of the BLOB column data in each row. Such a column should be named:

```
<Column Name>_ContentType
```

If the web server finds a column with this name, it will use the contents of the column as the **Content-Type** header when returning the BLOB data in a BLOB column request.

If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: GET

HTTP Response Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/example/albums
```

Example Response Content:

```
{
   "Columns": [{ "Name": "ID",
                 "Type": 1,
                 "Length": 38,
                 "Scale": null },
               { "Name": "Title",
                 "Type": 1,
                 "Length": 60,
                 "Scale": null },
               { "Name": "Artist",
                 "Type": 1,
                 "Length": 40,
                 "Scale": null },
               { "Name": "SortArtist",
                 "Type": 1,
                 "Length": 40,
                 "Scale": null },
               { "Name": "Year",
                 "Type": 3,
                 "Length": null,
                 "Scale": null },
               { "Name": "Label",
                 "Type": 1,
                 "Length": 40,
                 "Scale": null },
               { "Name": "CoverArt",
                 "Type": 8,
                 "Length": null,
                 "Scale": null },
               { "Name": "CoverArt_ContentType",
                 "Type": 1,
                 "Length": 40,
                 "Scale": null }]
}
```


HTTP Response: 200 on success or 500 on error

Get DataSet Parmeters Request

The get dataset parameters request enumerates all of the parameters in the dataset and database specified in the URL. The column information returned includes the name and type. The parameter types that can be returned are:

Column Type	Description
0	Unknown type
1	String
2	Boolean
3	Integer
4	Float
5	Date
6	Time
7	Date/Time
8	BLOB
9	CLOB (non-browser clients only)

If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: GET

HTTP Response Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/example/tracks/params
```

Example Response Content:

```
{
   "Params": [{ "Name": "AlbumID",
                "Type": 1 }]
}
```

HTTP Response: 200 on success or 500 on error

Get DataSet Rows Request

The get dataset rows request returns the rows in the dataset and database specified in the URL. Any input parameters to be used with the **Select** command for the dataset should be specified as URL parameters. Each column type in the JSON response content should be formatted according to the following rules:

Column Type	Description
String CLOB (non-browser clients only) BLOB	Enclose non-**null** values in double quotes.
Boolean	Specify **true** or **false** literals for non-**null** values.
Integer	Specify any valid integer value (positive or negative) for non-**null** values.
Float	Specify any valid floating-point value for non-**null** values. If not **null**, the incoming value must use the period (.) decimal separator if it contains fractional digits.
Date Time Date/Time	Specify any valid integer value (positive or negative) for non-**null** values. If not **null**, the incoming value represents the number of milliseconds since midnight on January 1, 1970.

If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: GET

HTTP Response Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/example/albums/data?ID=0201c645-c8fe-46b0-b785-815a617a1136
```

Example Response Content:

```
{
   "Rows": [{ "ID": "0201c645-c8fe-46b0-b785-815a617a1136",
              "Title": "August and Everything After",
              "Artist": "Counting Crows",
              "SortArtist": "Counting Crows",
              "Year": 1993,
              "Label": "BMG Direct Marketing, Inc.",
              "CoverArt":
      "?column=CoverArt&ID=0201c645-c8fe-46b0-b785-815a617a1136",
              "CoverArt_ContentType": "image\/jpeg" }]
}
```

HTTP Response: 200 on success or 500 on error

Get DataSet BLOB Column Request

The get dataset BLOB column request returns the data in the column, dataset, and database specified in the URL. Unlike the database and dataset names, the BLOB column name is specified in a **column** parameter instead of in the URL. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: GET

HTTP Response Content Type: Determined by the <Column Name>_ContentType column, if present, and application/octet-stream if not

Example Request:

```
https://localhost/example/albums/data?column=CoverArt&ID=0201c645-c8fe-46b0-b
      785-815a617a1136
```

HTTP Response: 200 on success or 500 on error

Commit Database Transaction Request

The commit database transaction request commits a transaction for the database specified in the URL using the transaction operations in the included JSON content. The operation rowset IDs in the included JSON content are special internal identifiers used by client applications to update any generated row values using the response content (see below). The following are the operation types supported in the included JSON content, as well as how they affect the structure of the included JSON content:

Operation Type	Description

1	Insert - the beforerow value will be **null** and the afterrow value will contain the row data for the inserted row.
2	Update - the beforerow value will contain the row data for the row before the update, and the afterrow value will contain the row data for the row after the update.
3	Delete - the beforerow value will contain the row data for the row before the deletion, and the afterrow value will be **null**.

The response to the commit database transaction request is a set of rows with a rowset ID, row ID, and row data, and will be used to update any inserted or updated rows in the client application with any newly-generated values that were created during the insert or update operation. For example, if an underlying table in the dataset contains an identity column that is assigned an identifier during row inserts, then the response content will contain the identifier.

> **Note**
> This functionality depends upon the ability of the database engine to use output parameters to capture generated row values. For example, the ElevateDB database engine supports using output parameters with INSERT statements to capture generated row values, but SQL Server will require that you use a script for your **Insert** dataset command that assigns the last generated identity value to an output parameter.

If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8
HTTP Response Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/example
```

Example Request Content:

```
{
   "Operations": [{ "DataSet": "Orders",
                    "RowSet": 2,
                    "Operation": 2,
                    "BeforeRow": { "EWBRowID": 4,
                                   "OrderNo": 1006,
                                   "CustNo": 1380,
                                   "SaleDate": 1497398400000,
                                   "ShipDate": 594907200000,
                                   "EmpNo": 46,
                                   "ShipVIA": "Emery",
                                   "PO": "P101324",
                                   "Terms": "FOB",
                                   "PaymentMethod": "Visa",
                                   "ItemsTotal": 31987,
                                   "TaxRate": 0,
                                   "Freight": 0,
                                   "AmountPaid": 0 },
                    "AfterRow": { "EWBRowID": 4,
```

```
                                    "OrderNo": 1006,
                                    "CustNo": 1380,
                                    "SaleDate": 1497398400000,
                                    "ShipDate": 594907200000,
                                    "EmpNo": 110,
                                    "PO": "P101324",
                                    "Terms": "FOB",
                                    "PaymentMethod": "Visa",
                                    "ItemsTotal": 31987,
                                    "TaxRate": 0,
                                    "Freight": 0,
                                    "AmountPaid": 0 } }]
}
```

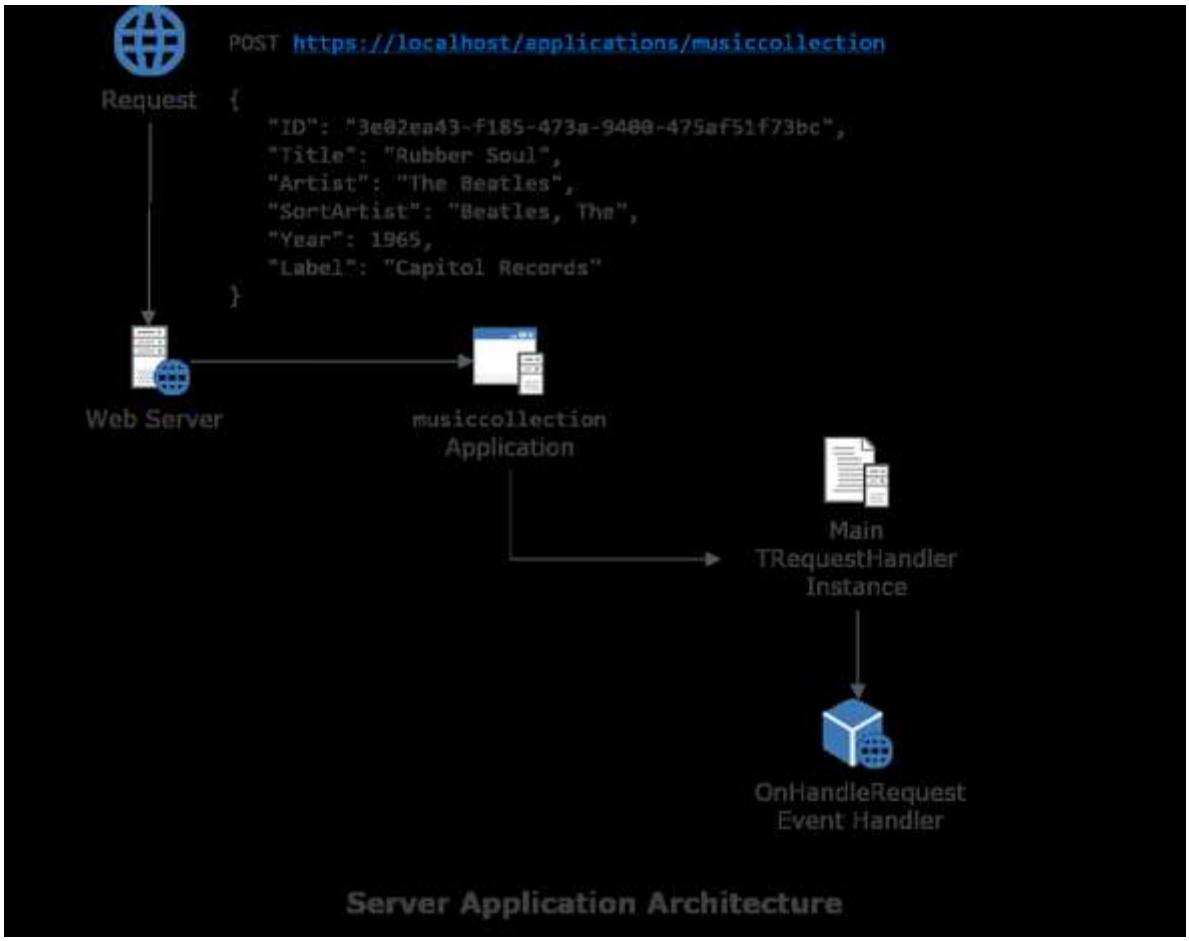
Example Response Content:

```
{
   "Rows": [{ "RowSet": 2,
              "EWBRowID": 4,
              "Row": {  } }]

}
```

HTTP Response: 200 on success or 500 on error

2.11 Web Server Applications

The web server includes an interpreter and runtime that allows authenticated users to make requests to server applications. Web server requests are routed to server applications using the following architecture:



Please see the Web Server Request Handling topic for more information on how requests are structured and routed by the web server.

Application Format

A server application uses a custom compressed binary format that bundles the application source units referenced during compilation into a single file with a .wba extension. When a server application is first required by the web server, the application is loaded and compiled into an in-memory runtime format before being executed. This runtime format will be used for executing the code in the server application until the application is unloaded.

Native External Modules

Native external modules are libraries that expose native type and function/procedures to server applications. The only native language support currently available is Delphi from Embarcadero Technologies. The support units for creating native external modules can be found in the **Elevate Web Builder Modules** installations available for Delphi XE2 or higher from the Elevate Software web site.

Note
Both native external modules and native server modules are libraries, but native server modules are self-contained native applications that are built specifically to handle incoming web server requests in a manner similar to server applications. Please see the Web Server Native Modules topic for more information on native server modules.

The following is the main unit of one of the example native external module library projects included with the **Elevate Web Builder Modules** installations:

```
unit extnamespacemain;

interface

{ You need to include the ewbextmodule unit in the uses clause }

uses SysUtils, ewbextmodule;

type

   { Here we define the various types used in the namespace that we are
     exposing to the Elevate Web Builder interpreter, including the namespace
     class itself }

   TMyNamespaceType = (ntNone,ntInternal,ntLibrary);

   { You can only register one namespace per module, and the namespace
     class must descend from the TEWBExternalModuleNamespace class.
     In addition, any properties that you wish to be visible to the
     Elevate Web Builder interpreter must be included in the published
     scope.  Any public methods will automatically be visible in the
     Elevate Web Builder interprepter because the ancestor
     TEWBExternalModuleNamespace class is marked with the
     $METHODINFO ON compiler directive }

   TMyExternalModuleNamespace = class(TEWBExternalModuleNamespace)
      private
         FID: Integer;
         FValue: String;
         FNamespaceType: TMyNamespaceType;
      public
         procedure TestProcedure(ID: Integer;
                                 const Value: String;
                                 NamespaceType: TMyNamespaceType);
         function TestFunction(ID: Integer): TMyNamespaceType;
      published
         property ID: Integer read FID write FID;
         property Value: String read FValue write FValue;
         property NamespaceType: TMyNamespaceType read FNamespaceType
                                                  write FNamespaceType;
      end;

implementation

{ This variable will hold the global instance of the namespace class instance
      }
var
   MyExternalModuleNamespace: TMyExternalModuleNamespace;

procedure TMyExternalModuleNamespace.TestProcedure(ID: Integer;
                                                   const Value: String;
```

```
                                                    NamespaceType:
      TMyNamespaceType);
begin
   FID:=ID;
   FValue:=Value;
   FNamespaceType:=NamespaceType;
end;

function TMyExternalModuleNamespace.TestFunction(ID: Integer):
      TMyNamespaceType;
begin
   if (ID=FID) then
      Result:=FNamespaceType
   else
      Result:=ntNone;
end;

{ Here we create the namespace class instance and assign it a type alias
  for the TMyNamespaceType enumerated type so that the Elevate Web Builder
  interpreter can refer to the type using a more normalized name }

initialization
   MyExternalModuleNamespace:=TMyExternalModuleNamespace.Create;
   RegisterExternalModule(MyExternalModuleNamespace);
   RegisterExternalTypeAlias('TMyNamespaceType','TNamespaceType');
finalization
   UnregisterExternalModule;
   FreeAndNil(MyExternalModuleNamespace);
end.
```

Once a native external module has been built, it can be included as an external file in a server application project to ensure that it is deployed along with the server application. Please see the Using the Project Manager topic for more information on including external files with projects.

Note
Native external modules are loaded once when the first external declaration is bound during server application compilation, and remain loaded until the application is unloaded. Therefore, you will want to institute a versioning scheme in the native external module file names in order to avoid a situation where a deployment fails due to a native external module still being loaded in the web server process space, resulting in the web server being unable to overwrite the native external module file with the new version.

In order to actually reference any of the exposed declarations in a native external module, you must first create an external interface unit for the native external module. Please see the Creating a New External Module Interface Unit for more information on how to automatically generate an external interface unit for a native external module using the IDE.

The following is the external interface unit generated for the above native external module:

```
unit extnamespace;

interface

type

   { Enumerated Types }

   external TNamespaceType = (ntNone, ntInternal, ntLibrary) module
      extnamespace;
```

```
   { Routines }

   external procedure TestProcedure(ID: Integer; const Value: String;
      NamespaceType: TNamespaceType) module extnamespace;
   external function TestFunction(ID: Integer): TNamespaceType module
      extnamespace;

   { Variables }

   var
      external ID: Integer module extnamespace;
      external Value: String module extnamespace;
      external NamespaceType: TNamespaceType module extnamespace;

implementation
end.
```

Please see the External Interfaces topic for more information on the structure of external interface units and declarations.

Application Deployment

Server applications can be deployed using the administration API in the web server. The IDE uses this API to deploy server applications and any associated external files to a specified path within the **Applications** virtual resource on the web server. Please see the Deploying a Project and Web Server Administration API - Files topics for more information.

Application Installation

Server application deployment is insufficient for allowing a server application to be accessible to authenticated users, and an application must first be installed before incoming web server requests can be routed to the application. The IDE allows the user to install a server application using the server manager.

> **Note**
> Server applications, like all server objects in the web server, are versioned. Once a server application has been installed, the underlying application .wba file can be updated while the web server is running without being required to be installed again.

Please see the Using the Server Manager topic for more information on using the server manager in the IDE and the Web Server Administration API - Applications topic for more information on installing and uninstalling applications.

Application Execution

When an incoming request is routed to a particular server application and the application code is executed, an execution environment is obtained. These execution environments are cached for each application in order to minimize application startup times.

The server application memory is initialized and finalized for every server application execution. During initialization, a global TApplication component instance is created along with the main TRequestHandler for the server application. The incoming web server TWebServerRequest instance is then routed to the TRequestHandler instance's OnHandleRequest event handler, if one is defined. The event handler code can then handle the request and send a response to the request using one of the following TWebServerRequest methods:

Method	Description

SendContentHeader	Sends a response for a HEAD request.
SendCustomContentHeader	Sends a custom content response for a HEAD request.
SendContent	Sends a UTF-8-encoded text response with a **Content-Type** header of "text/html; charset=utf-8" along with an optional status code and message.
SendCustomContent	Sends a UTF-8-encoded text response with a custom content type, encoding, and disposition.
SendRedirect	Sends a redirect response to a new URL along with an optional UTF-8-encoded text response with a **Content-Type** header of "text/html; charset=utf-8". By default, the redirect HTTP status code is 302, but you can specify a different status code.
SendError	Sends a status code and message along with an optional UTF-8-encoded text response with a **Content-Type** header of "text/plain; charset=utf-8".
SendContentStream	Sends a UTF-8-encoded text stream response with a **Content-Type** header of "text/html; charset=utf-8" along with an optional status code and message.
SendCustomContentStream	Sends a binary stream response with a custom content type, encoding, and disposition.
SendContentFile	Sends a file response with a custom content type, encoding, and disposition. **Note** If the content type is not specified, then the web server will attempt to determine the content type based upon the file extension.

Warning
When debugging server applications in the IDE, the web server will automatically send a response if none of the above methods are called by the application and execution of the application terminates. However, when executing server applications outside of a debugging context, the web server will not automatically send a respons and this can cause the client application to hang and, eventually, time out waiting on a response that will never arrive.

You can encode a stream response using the TStream Compress method. However, make sure that you set the content encoding parameter properly to ensure that the client application can decompress the content with errors. Use the TWebServerRequest instance's AcceptsContentEncoding method to determine whether to use a particular type of content encoding.

2.12 Web Server Applications API

The server applications API requests use a REST-ful style that is structured as follows:

```
<Origin>/applications/<Application>?<Parameters>

<Origin> = <Protocol>://<Domain>[:<Port>]

<Application> = Server application name

<Parameters> = URL parameters in key=value form and an
               ampersand (&) between parameters
```

Note
The "applications" resource name is the default resource name for server application requests. The application resource name is configurable in the web server. Please make sure that the resource name that you are using matches the proper resource name on the target web server.

Any content included with server application API requests, or returned as a response to the request, should/will be formatted as JSON content. The date/time format used in the JSON content is equivalent to a raw JavaScript Date value: an integer value representing the number of milliseconds since 1 January 1970 UTC.

Note
Any forward slashes present in any JSON content returned as a response by the server application API will be escaped so that the JSON content can be included in HTML without issues.

Execute Application Request

The execute application request executes the application specified in the URL, passing in any subsequent path information or parameters included in the rest of the URL. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

Note
The debugger executes this request with a special **mode** parameter set to **debug**. You should not use this parameter because doing so will cause the client application to hang if it doesn't handle the debugger initialization requests and responses properly.

HTTP Method: Any valid HTTP method

HTTP Request Content Type: Any valid content type
HTTP Response Content Type: Any valid content type

Example Request:

```
PATCH
      https://localhost/applications/musiccollection/3e02ea43-f185-473a-9400-475af5
```

```
      1f73bc/5
```

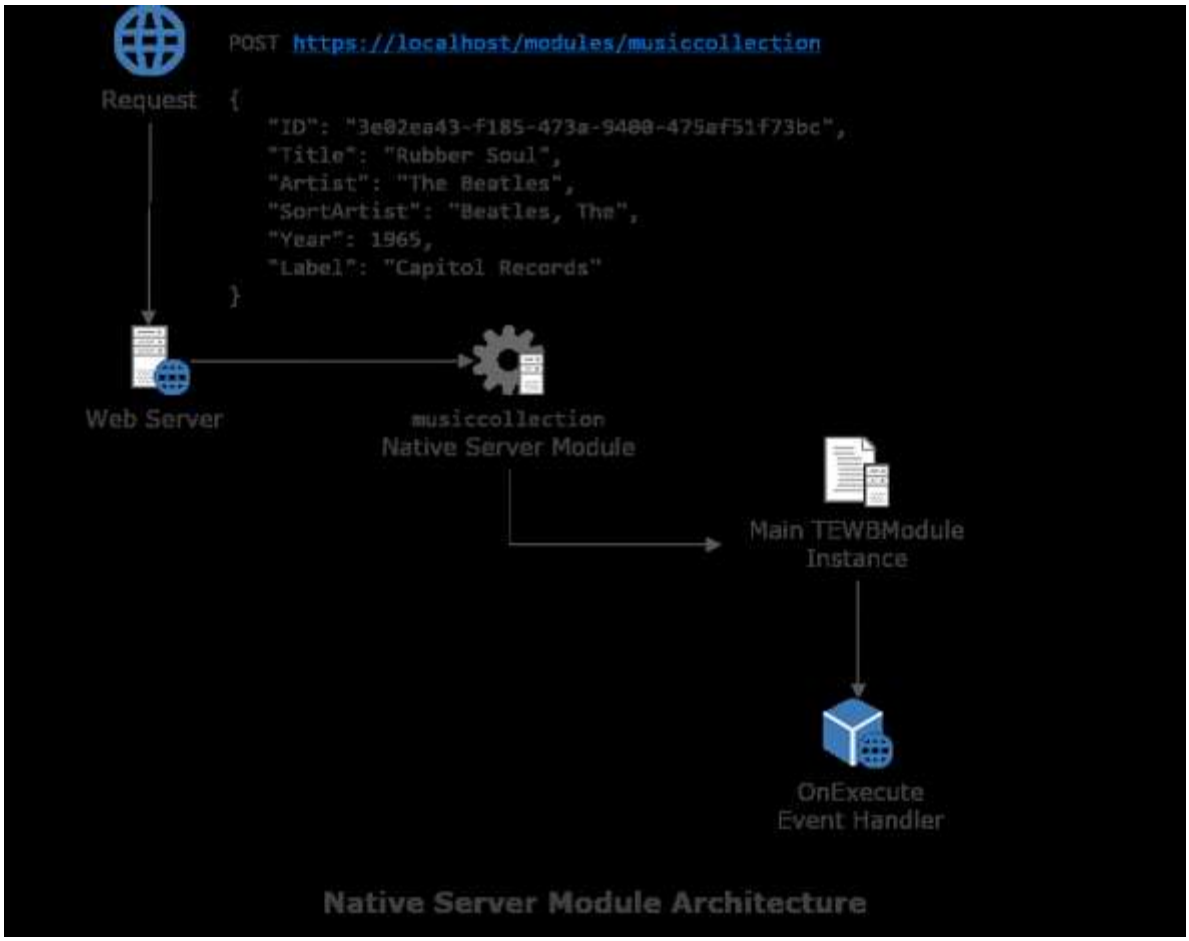
Example Request Content:

```
{
   "MediaID": "SJSuBMrDMpo"
}
```

HTTP Response: Any valid HTTP status code

2.13 Web Server Native Modules

The web server includes a native server module manager that allows authenticated users to make requests to native server modules. Web server requests are routed to native server modules using the following architecture:



Please see the Web Server Request Handling topic for more information on how requests are structured and routed by the web server.

Native Module Format

A native server module is a library that can handle incoming web server requests. When a native server module is required by the web server, the server module is loaded before being executed.

For information on creating native server modules, please refer to the product-specific manual that accompanies the **Elevate Web Builder 3 Modules** product installation. You can download and install the **Elevate Web Builder 3 Modules** installation for your specific product using the following link:

Elevate Web Builder Downloads

Modules can only be created and built using Embarcadero RAD Studio and Delphi XE2 or higher.

Native Module Deployment

Native server modules can be deployed using the administration API in the web server. The IDE uses this API to

deploy server modules to a specified path within the **Modules** virtual resource on the web server. Please see the Using the Server Manager and Web Server Administration API - Files topics for more information.

Native Module Installation

Native server module deployment is insufficient for allowing a server module to be accessible to authenticated users, and a server module must first be installed before incoming web server requests can be routed to the module. The IDE allows the user to install a native server module using the server manager.

> **Note**
> Native server modules, like all server objects in the web server, are versioned. However, once a native server module has been installed, the underlying library cannot be reliably updated while the web server is running because, at any given moment in time, the server module library could be loaded in the web server process and, therefore, cannot be overwritten. For this reason, you should consider using a versioning scheme with native server module file names and increment the version and/or build number in the native server module file name. You will then need to upload the new native server module to the web server and re-install the server module using the new file name. This will cause the web server to create a new version of the native server module and remove the older version once all instances of the server module are unloaded by any executing threads in the web server.

Please see the Using the Server Manager topic for more information on using the server manager in the IDE and the Web Server Administration API - Modules topic for more information on installing and uninstalling native server modules.

Native Module Execution

When an incoming request is routed to a particular native server module and the library code is executed, an execution environment is obtained. These execution environments are cached for each server module in order to minimize module startup times.

The native server module memory is initialized once when the library is loaded and finalized when the library is unloaded. If any global initialization is required for a database engine or other support code, it should be placed in the initialization section of a source unit in the Delphi native server module project. Similarly, any global teardown code should be place in the finalization section of a source unit in the Delphi native server module project.

A global TEWBModule component instance is created for every incoming request to the web server. The incoming web server TEWBWebServerRequest instance is then routed to the TEWBModule instance's OnExecute event handler, if one is defined. The event handler code can then handle the request and send a response to the request using one of the following TEWBWebServerRequest methods:

Method	Description

SendContentHeader	Sends a response for a HEAD request.
SendCustomContentHeader	Sends a custom content response for a HEAD request.
SendContent	Sends a UTF-8-encoded text response with a **Content-Type** header of "text/html; charset=utf-8" along with an optional status code and message.
SendCustomContent	Sends a UTF-8-encoded text response with a custom content type, encoding, and disposition.
SendRedirect	Sends a redirect response to a new URL along with an optional UTF-8-encoded text response with a **Content-Type** header of "text/html; charset=utf-8". By default, the redirect HTTP status code is 302, but you can specify a different status code.
SendError	Sends a status code and message along with an optional UTF-8-encoded text response with a **Content-Type** header of "text/plain; charset=utf-8".
SendContentStream	Sends a UTF-8-encoded text stream response with a **Content-Type** header of "text/html; charset=utf-8" along with an optional status code and message.
SendCustomContentStream	Sends a binary stream response with a custom content type, encoding, and disposition.

Warning
If none of the above methods are called by the native server module and execution of the module terminates, the web server will not automatically send a response. This can cause the client application to hang and, eventually, time out waiting on a response that will never arrive.

2.14 Web Server Native Modules API

The server native modules API requests use a REST-ful style that is structured as follows:

```
<Origin>/modules/<Module>?<Parameters>

<Origin> = <Protocol>://<Domain>[:<Port>]

<Module> = Native server module name

<Parameters> = URL parameters in key=value form and an
               ampersand (&) between parameters
```

Note
The "modules" resource name is the default resource name for native server module requests. The module resource name is configurable in the web server. Please make sure that the resource name that you are using matches the proper resource name on the target web server.

Any content included with native server module API requests, or returned as a response to the request, should/will be formatted as JSON content. The date/time format used in the JSON content is equivalent to a raw JavaScript Date value: an integer value representing the number of milliseconds since 1 January 1970 UTC.

Note
Any forward slashes present in any JSON content returned as a response by the native server module API will be escaped so that the JSON content can be included in HTML without issues.

Execute Module Request

The execute module request executes the native server module specified in the URL, passing in any subsequent path information or parameters included in the rest of the URL. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: Any valid HTTP method

HTTP Request Content Type: Any valid content type
HTTP Response Content Type: Any valid content type

Example Request:

```
PATCH
      https://localhost/modules/musiccollection/3e02ea43-f185-473a-9400-475af51f73b
      c/5
```

Example Request Content:

```
{
```

```
    "MediaID": "SJSuBMrDMpo"
}
```

HTTP Response: Any valid HTTP status code

2.15 Web Server Administration

The web server provides an administration API that allows any client application that can make HTTP requests to remotely administer the web server. This includes updating privileges, roles, users, databases, server applications, and native server modules, as well as uploading static content, managing static content folders, and retrieving server statistics.

Currently, the IDE is the only client application capable of providing an interface to the entire server administration API. A future update to the product will include this functionality in the TServerSession component included with the component library. Please see the Using the Server Manager topic for more information on using the server manager in the IDE to interactively administer the web server.

2.16 Web Server Administration API

The administration API requests use a remote procedure call style that is structured as follows:

```
<Origin>/administration?method=<Method>

<Origin> = <Protocol>://<Domain>[:<Port>]

<Method> = Administration method name
```

Note
The "administration" resource name is the default resource name for administration requests. The administration resource name is configurable in the web server. Please make sure that the resource name that you are using matches the proper resource name on the target web server.

Any content included with administration API requests, or returned as a response to the request, should/will be formatted as JSON content. The date/time format used in the JSON content is equivalent to a raw JavaScript Date value: an integer value representing the number of milliseconds since 1 January 1970 UTC.

Note
Any forward slashes present in any JSON content returned as a response by the administration API will be escaped so that the JSON content can be included in HTML without issues.

Due to its size, the API reference for the server administration has been divided into several different topics:

Web Server Administration API - Privileges
Web Server Administration API - Roles
Web Server Administration API - Users
Web Server Administration API - Databases
Web Server Administration API - Modules
Web Server Administration API - Applications
Web Server Administration API - Files
Web Server Administration API - Status

2.17 Web Server Administration API - Privileges

Add Privilege Request

The add privilege request adds the privilege provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=addprivilege
```

Example Request Content:

```
{
   "Name": "ViewCustomerInvoices",
   "Description": "View customer invoices"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Update Privilege Request

The update privilege request updates the privilege provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=updateprivilege
```

Example Request Content:

```
{
   "Name": "ViewCustomerInvoices",
   "Description": "View customer invoices"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Remove Privilege Request

The remove privilege request removes the privilege provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=removeprivilege
```

Example Request Content:

```
{
   "Name": "ViewCustomerInvoices"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Get Privileges Request

The get privileges request enumerates all of the privileges defined on the web server. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: GET

HTTP Response Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=getprivileges
```

Example Response Content:

```
{
   "Privileges": [
      { "Name": "AddDatabase",
        "Description": "Add a new database" },
      { "Name": "AddDataSet",
        "Description": "Add a new dataset" },
      { "Name": "AddDataSetCommand",
        "Description": "Add a new dataset command" },
      { "Name": "AddPrivilege",
        "Description": "Add a new privilege" }]
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

2.18 Web Server Administration API - Roles

Add Role Request

The add role request adds the role provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=addrole
```

Example Request Content:

```
{
   "Name": "CustomerService",
   "Description": "Customer Service Role"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Update Role Request

The update role request updates the role provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=updaterole
```

Example Request Content:

```
{
   "Name": "CustomerService",
   "Description": "Customer Service Role"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Remove Role Request

The remove role request removes the role provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=removerole
```

Example Request Content:

```
{
   "Name": "CustomerService"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Activate Role Request

The activate role request changes the status of the role provided in the included JSON content to active. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=activaterole
```

Example Request Content:

```
{
   "Name": "CustomerService"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Deactivate Role Request

The deactivate role request changes the status of the role provided in the included JSON content to inactive. If there

is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=deactivaterole
```

Example Request Content:

```
{
   "Name": "CustomerService"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Grant Role Privileges Request

The grant role privileges request grants a set of privileges to the role provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=grantroleprivileges
```

Example Request Content:

```
{
   "Name": "CustomerService",
   "Privileges": ["AddCustomer", "UpdateCustomer", "ViewInvoices"]
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Revoke Role Privileges Request

The revoke role privileges request revokes a set of privileges from the role provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=revokeroleprivileges
```

Example Request Content:

```
{
   "Name": "CustomerService",
   "Privileges": ["AddCustomer", "UpdateCustomer", "ViewInvoices"]
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Get Role Privileges Request

The get role privileges request enumerates all of the privileges assigned to the role provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: GET

HTTP Response Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=getroleprivileges
```

Example Request Content:

```
{
   "Name": "CustomerService"
}
```

Example Response Content:

```
{
   "Privileges": ["AddCustomer", "UpdateCustomer", "ViewInvoices"]
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Get Roles Request

The get roles request enumerates all of the roles defined on the web server. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: GET

HTTP Response Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=getroles
```

Example Response Content:

```
{
   "Roles": [
      { "Name": "Administrators",
        "Active": true,
        "Description": "Administrators Role",
        "Privileges": [{ "Name": "AddDatabase" },
                       { "Name": "AddDataSet" },
                       { "Name": "AddDataSetCommand" },
                       { "Name": "AddPrivilege" },
                       { "Name": "AddRole" },
                       { "Name": "AddSystemEvent" }] },
      { "Name": "CustomerService",
        "Active": true,
        "Description": "Customer Service",
        "Privileges": [{ "Name": "AddCustomer" },
                       { "Name": "UpdateCustomer" },
                       { "Name": "ViewInvoices" }] },
      { "Name": "Public",
        "Active": true,
        "Description": "Public Role",
        "Privileges": [] }]
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

2.19 Web Server Administration API - Users

Add User Request

The add user request adds the user provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=adduser
```

Example Request Content:

```
{
   "Name": "JohnSmith",
   "FullName": "John Smith",
   "EmailAddress": "john.smith@company.com",
   "LoginStartTime": -2209161600000,
   "LoginEndTime": -2209075201000
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

> **Note**
> New users are always created in an inactive state, and must be activated using the Activate User Request below in order to be available for use with the web server.

Update User Request

The update user request updates the user provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=updateuser
```

Example Request Content:

```
{
   "Name": "JohnSmith",
   "FullName": "John Smith",
   "EmailAddress": "john.smith@company.com",
   "LoginStartTime": -2209161600000,
   "LoginEndTime": -2209075201000
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Remove User Request

The remove user request removes the user provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=removeuser
```

Example Request Content:

```
{
   "Name": "JohnSmith"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Activate User Request

The activate user request changes the status of the user provided in the included JSON content to active. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=activateuser
```

Example Request Content:

```
{
   "Name": "JohnSmith"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Deactivate User Request

The deactivate user request changes the status of the user provided in the included JSON content to inactive. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=deactivateuser
```

Example Request Content:

```
{
   "Name": "JohnSmith"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Lock User Request

The lock user request locks the user provided in the included JSON content so that the user cannot authenticate with the web server until the provided date/time. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=lockuser
```

Example Request Content:

```
{
   "Name": "JohnSmith",
   "LockReleaseTime": 1607392690982
```

```
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Unlock User Request

The unlock user request unlocks the user provided in the included JSON content so that the user can authenticate with the web server. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=unlockuser
```

Example Request Content:

```
{
   "Name": "JohnSmith"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Set User Password Request

The set user password request changes the password for the user provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=setuserpassword
```

Example Request Content:

```
{
   "Name": "JohnSmith",
   "Password": "HelloWorld123456"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Grant User Roles Request

The grant user roles request grants a set of roles to the user provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=grantuserroles
```

Example Request Content:

```
{
   "Name": "JohnSmith",
   "Roles": ["CustomerService", "Public"]
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Revoke User Roles Request

The revoke user roles request revokes a set of roles from the user provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=revokeuserroles
```

Example Request Content:

```
{
   "Name": "JohnSmith",
   "Roles": ["CustomerService", "Public"]
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Get User Roles Request

The get user roles request enumerates all of the roles assigned to the user provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: GET

HTTP Request Content Type: application/json; charset=utf-8
HTTP Response Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=getuserroles
```

Example Request Content:

```
{
   "Name": "JohnSmith"
}
```

Example Response Content:

```
{
   "Roles": ["CustomerService", "Public"]
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Get Users Request

The get users request enumerates all of the users defined on the web server. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: GET

HTTP Response Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=getusers
```

Example Response Content:

```
{
   "Users": [
      { "Name": "Administrator",
        "Active": true,
        "FullName": "Administrator User",
```

```
        "EmailAddress": "",
        "LoginStartTime": -2209161600000,
        "LoginEndTime": -2209075201000,
        "Locked": false,
        "LockReleaseTime": 0,
        "Roles": [{ "Name": "Administrators" },
                  { "Name": "Public" }] },
      { "Name": "Anonymous",
        "Active": true,
        "FullName": "Anonymous User",
        "EmailAddress": "",
        "LoginStartTime": -2209161600000,
        "LoginEndTime": -2209075201000,
        "Locked": false,
        "LockReleaseTime": 0,
        "Roles": [{ "Name": "Public" }] },
      { "Name": "JohnSmith",
        "Active": true,
        "FullName": "John Smith",
        "EmailAddress": "john.smith@company.com",
        "LoginStartTime": -2209161600000,
        "LoginEndTime": -2209075201000,
        "Locked": false,
        "LockReleaseTime": 0,
        "Roles": [{ "Name": "CustomerService" },
                  { "Name": "Public" }] }]
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

2.20 Web Server Administration API - Databases

The engine type provided in the included JSON content for several of the database requests should be one of the following:

Type	Engine
0	ElevateDB
1	DBISAM
2	ADO/ODBC

In addition, several of the database requests also allow specifying database connection properties that are specific to the type of database engine.

Test Database Request

The test database request tests the database provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=testdatabase
```

Example Request Content:

```
{
   "Name": "Invoicing",
   "Description": "Invoicing Database",
   "EngineType": 0,
   "Properties": { "CharacterSet": 1,
                   "DatabaseName": "Test",
                   "LoginPassword": "DBUser123456",
                   "LoginUser": "DBUser",
                   "RemoteAddress": "127.0.0.1",
                   "SessionDescription": "EWB Session",
                   "SessionName": "EWB",
                   "SessionType": 1 }
}
```

Example Response Content:

```
{
   "Result": true,
   "Message": ""
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Add Database Request

The add database request adds the database provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=adddatabase
```

Example Request Content:

```
{
   "Name": "Invoicing",
   "Description": "Invoicing Database",
   "EngineType": 0,
   "Properties": { "CharacterSet": 1,
                   "DatabaseName": "Test",
                   "LoginPassword": "DBUser123456",
                   "LoginUser": "DBUser",
                   "RemoteAddress": "127.0.0.1",
                   "SessionDescription": "EWB Session",
                   "SessionName": "EWB",
                   "SessionType": 1 }
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Update Database Request

The update database request updates the database provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=updatedatabase
```

```
{
   "Name": "Invoicing",
```

```
   "Description": "Invoicing Database",
   "EngineType": 0,
   "Properties": { "CharacterSet": 1,
                   "DatabaseName": "Test",
                   "LoginPassword": "DBUser123456",
                   "LoginUser": "DBUser",
                   "RemoteAddress": "127.0.0.1",
                   "SessionDescription": "EWB Session",
                   "SessionName": "EWB",
                   "SessionType": 1 }
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Rename Database Request

The remame database request renames the database provided in the included JSON content to a new name. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=renamedatabase
```

Example Request Content:

```
{
   "Name": "Invoicing",
   "NewName": "InvoicingDB"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Remove Database Request

The remove database request removes the database provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=removedatabase
```

Example Request Content:

```
{
   "Name": "Invoicing"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Get Database Privileges Request

The get database privileges request retrieves the access privilege assigned to the database provided in the included JSON content. If the access privilege has not been specified, then the database is accessible to all users. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: GET

HTTP Request Content Type: application/json; charset=utf-8
HTTP Response Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=getdatabaseprivileges
```

Example Request Content:

```
{
   "Name": "Invoicing"
}
```

Example Response Content:

```
{
   "Privilege": "InvoicingData"]
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Set Database Privileges Request

The set database privileges request assigns the access privilege for the database provided in the included JSON content. If the access privilege is not specified, then the database will be accessible to all users. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=setdatabaseprivileges
```

Example Request Content:

```
{
   "Name": "Invoicing",
   "Privilege": "InvoicingData"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Get Database Tables Request

The get database tables request enumerates all of the tables available in the database provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

> **Note**
> The successful connection to the underlying database is required for this request to succeed. If the underlying database is not available for any reason, then an error indicating the reason will be returned.

HTTP Method: GET

HTTP Request Content Type: application/json; charset=utf-8
HTTP Response Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=getdatabasetables
```

Example Request Content:

```
{
   "Name": "Invoicing"
}
```

Example Response Content:

```
{
   "Tables": ["Customer", "InvoiceHeader", "InvoiceDetail"]
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Get Databases Request

The get databases request enumerates all of the databases defined on the web server. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: GET

HTTP Response Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=getdatabases
```

Example Response Content:

```
{
   "Databases": [
      { "Name": "Invoicing",
        "Description": "Invoicing Database",
        "EngineType": 0,
        "Properties": { "CharacterSet": 1,
                        "DatabaseName": "Test",
                        "LoginPassword": "DBUser123456",
                        "LoginUser": "DBUser",
                        "RemoteAddress": "127.0.0.1",
                        "SessionDescription": "EWB Session",
                        "SessionName": "EWB",
                        "SessionType": 1 },
        "DataSets": [{ "Name": "Customers",
                       "Description": "Customers DataSet",
                       "LocalizeDateTimes": false,
                       "Commands": [{ "Name": "Select",
                                      "Description": "", "SQL": "SELECT *
      FROM customer",
                                      "Logged": false,
                                      "Privilege": "" },
                                    { "Name": "Insert",
                                      "Description": "", "SQL": "INSERT INTO
      customer VALUES (:CustNo, :InvoiceNo, :InvoiceDate)",
                                      "Logged": false,
                                      "Privilege": "" }] }] }]
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Add DataSet Request

The add dataset request adds the dataset provided in the included JSON content. The LocalizeDateTimes property specifies whether date-time column values should be localized when read or written from the dataset. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=adddataset
```

Example Request Content:

```
{
   "Database": "Invoicing",
   "Name": "Customers",
   "Description": "Customers DataSet",
   "LocalizeDateTimes": false
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Update DataSet Request

The update dataset request updates the dataset provided in the included JSON content. The LocalizeDateTimes property specifies whether date-time column values should be localized when read or written from the dataset. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=updatedataset
```

```
{
   "Database": "Invoicing",
   "Name": "Customers",
   "Description": "Customers DataSet",
   "LocalizeDateTimes": false
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Rename DataSet Request

The remame dataset request renames the dataset provided in the included JSON content to a new name. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=renamedataset
```

Example Request Content:

```
{
   "Database": "Invoicing",
   "Name": "Customers",
   "NewName": "Customer"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Remove DataSet Request

The remove datasete request removes the dataset provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=removedataset
```

Example Request Content:

```
{
   "Database": "Invoicing",
   "Name": "Customers"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Generate DataSet Commands Request

The generate dataset commands request generates commands for the dataset provided in the included JSON content. The BaseTableName property specifies which base table to use for generating the commands, and the remaining properties specify which commands should be generated. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=generatedatasetcommands
```

Example Request Content:

```
{
   "Database": "Invoicing",
   "Name": "Customers",
   "BaseTableName": "Customer",
   "SelectCommand": true,
   "InsertCommand": true,
   "UpdateCommand": true,
   "DeleteCommand": true,
   "BLOBSelectCommands": true,
   "BLOBUpdateCommands": true
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Add DataSet Command Request

The add dataset command request adds the dataset command provided in the included JSON content. The Logged property specifies that any execution of the dataset command should be logged in the system log. Please see the Web Server Logging topic for more information on the system logging. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=adddatasetcommand
```

Example Request Content:

```
{
   "Database": "Invoicing",
   "DataSet": "Customers",
   "Name": "Select",
   "Description": "",
   "SQL": "SELECT * FROM customer",
   "Logged": false
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Update DataSet Command Request

The update dataset command request updates the dataset command provided in the included JSON content. The Logged property specifies that any execution of the dataset command should be logged in the system log. Please see the Web Server Logging topic for more information on the system logging. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=updatedatasetcommand
```

```
{
   "Database": "Invoicing",
   "DataSet": "Customers",
   "Name": "Select",
   "Description": "",
   "SQL": "SELECT * FROM customer",
   "Logged": false
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Rename DataSet Command Request

The remame dataset command request renames the dataset command provided in the included JSON content to a new name. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=renamedatasetcommand
```

Example Request Content:

```
{
   "Database": "Invoicing",
   "DataSet": "Customers",
   "Name": "Select",
   "NewName": "SelectCommand"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Remove DataSet Command Request

The remove dataset command request removes the dataset command provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=removedatasetcommand
```

Example Request Content:

```
{
   "Database": "Invoicing",
   "DataSet": "Customers",
   "Name": "Select"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Get DataSet Command Privileges Request

The get dataset command privileges request retrieves the access privilege assigned to the dataset command provided in the included JSON content. If the access privilege has not been specified, then the dataset command is accessible to all users. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: GET

HTTP Request Content Type: application/json; charset=utf-8
HTTP Response Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=getdatasetcommandprivileges
```

Example Request Content:

```
{
   "Database": "Invoicing",
   "DataSet": "Customers",
   "Name": "Select"
```

```
}
```

Example Response Content:

```
{
   "Privilege": ""
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Set DataSet Command Privileges Request

The set dataset command privileges request assigns the access privilege for the dataset command provided in the included JSON content. If the access privilege is not specified, then the dataset command will be accessible to all users. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=setdatasetcommandprivileges
```

Example Request Content:

```
{
   "Database": "Invoicing",
   "DataSet": "Customers",
   "Name": "Select",
   "Privilege": "ViewCustomers",
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

2.21 Web Server Administration API - Modules

Install Module Request

The install module request installs the native server module provided in the included JSON content. The module file name is a URL path that should begin with the virtual resource name **Modules** and may contain sub-folders, but must refer to an existing valid native server module file already present on the web server. Please see the Web Server Administration API - Files topic for more information on the virtual resource names used when managing files on the web server. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=installmodule
```

Example Request Content:

```
{
   "Name": "ContactForm",
   "FileName": "Modules\/contactform.dll"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Rename Module Request

The remame module request renames the native server module provided in the included JSON content to a new name. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=renamemodule
```

Example Request Content:

```
{
   "Name": "ContactForm",
   "NewName": "ContactFormModule"
```

```
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Uninstall Module Request

The uninstall module request uninstalls the native server module provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

> **Note**
> This request only uninstalls the module so that it no longer is a reachable endpoint for incoming web server requests. It does not remove the underlying native server module file from the web server. Please see the Web Server Administration API - Files topic for more information on deleting files present on the web server.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=uninstallmodule
```

Example Request Content:

```
{
   "Name": "ContactForm"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Get Module Privileges Request

The get module privileges request retrieves the execute privilege assigned to the native server module provided in the included JSON content. If the execute privilege has not been specified, then the native server module is accessible to all users. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: GET

HTTP Request Content Type: application/json; charset=utf-8
HTTP Response Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=getmoduleprivileges
```

Example Request Content:

```
{
   "Name": "ContactForm"
}
```

Example Response Content:

```
{
   "Privilege": "ExecuteContactForm"]
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Set Module Privileges Request

The set module privileges request assigns the execute privilege for the native server module provided in the included JSON content. If the execute privilege is not specified, then the native server module will be accessible to all users. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=setmoduleprivileges
```

Example Request Content:

```
{
   "Name": "ContactForm"
   "Privilege": "ExecuteContactForm"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Get Modules Request

The get modules request enumerates all of the native server modules installed on the web server. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: GET

Example Request:

```
https://localhost/administration?method=getmodules
```

Example Response Content:

```
{
   "Modules": [{ "Name": "ContactForm",
                 "FileName": "Modules\/contactform.dll",
                 "Privilege": "" }]
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

2.22 Web Server Administration API - Applications

Install Application Request

The install application request installs the server application provided in the included JSON content. The application file name is a URL path that should begin with the virtual resource name **Applications** and may contain sub-folders, but must refer to an existing valid server application file already present on the web server. Please see the Web Server Administration API - Files topic for more information on the virtual resource names used when managing files on the web server. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=installapplication
```

Example Request Content:

```
{
   "Name": "MusicCollection",
   "FileName": "Applications\/musiccollection.wba"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Rename Application Request

The remame application request renames the server application provided in the included JSON content to a new name. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=renameapplication
```

Example Request Content:

```
{
   "Name": "MusicCollection",
   "NewName": "MusicCollectionApplication"
```

```
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Uninstall Application Request

The uninstall application request uninstalls the server application provided in the included JSON content. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

> **Note**
> This request only uninstalls the application so that it no longer is a reachable endpoint for incoming web server requests. It does not remove the underlying server application file from the web server. Please see the Web Server Administration API - Files topic for more information on deleting files present on the web server.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=uninstallapplication
```

Example Request Content:

```
{
   "Name": "MusicCollection"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Get Application Privileges Request

The get application privileges request retrieves the execute privilege assigned to the server application provided in the included JSON content. If the execute privilege has not been specified, then the server application is accessible to all users. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: GET

HTTP Request Content Type: application/json; charset=utf-8
HTTP Response Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=getapplicationprivileges
```

Example Request Content:

```
{
   "Name": "MusicCollection"
}
```

Example Response Content:

```
{
   "Privilege": "ExecuteMusicCollection"]
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Set Application Privileges Request

The set application privileges request assigns the execute privilege for the server application provided in the included JSON content. If the execute privilege is not specified, then the server application will be accessible to all users. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=setapplicationprivileges
```

Example Request Content:

```
{
   "Name": "MusicCollection"
   "Privilege": "ExecuteMusicCollection"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Get Applications Request

The get applications request enumerates all of the server applications installed on the web server. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: GET

Example Request:

```
https://localhost/administration?method=getapplications
```

Example Response Content:

```
{
   "Applications": [{ "Name": "MusicCollection",
                "FileName": "Applications\/musiccollection.wba",
                "Privilege": "" }]
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

2.23 Web Server Administration API - Files

Any file names used in the file management portion of the administration API can refer to one of several virtual resource names:

Virtual Resource	Description
Content	Static content virtual resource
Modules	Native server modules virtual resource
Applications	Server applications virtual resource
Logs	Logs virtual resource

> **Note**
> These virtual resource names are only used with the administration API and should not to be confused with the resource names that are configured in the web server for request routing.

The virtual resource name should be the first portion of the file name path, and should use Unix/Web style path delimiters:

```
<Origin>/<Virtual Resource>[/<Folder>][/<File>]

<Origin> = <Protocol>://<Domain>[:<Port>]

<Virtual Resource> = Virtual resource name

                        Content
                        Modules
                        Applications
                        Logs

<Folder> = Folder name, if specified

<File> = File Name, if specified
```

File Modified Request

The file modified request returns whether the file provided in the included JSON content has been modified. The file name is a URL path that should begin with a virtual resource name and may contain sub-folders, but must refer to an existing valid file already present on the web server. The size and last modified properties are used to determine if the file has been modified. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: GET

HTTP Request Content Type: application/json; charset=utf-8
HTTP Response Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=filemodified
```

Example Request Content:

```
{
   "Name": "\/Content\/musiccollection.html",
   "Size": 173678,
   "LastModified": 1607220954332
}
```

Example Response Content:

```
{
   "Modified": false
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Upload File Request

The upload file request is a multi-part, mixed request that uploads the file provided in the included JSON content. The file name is a URL path that should begin with a virtual resource name and may contain sub-folders, but must refer to a valid path for the uploaded file. The offset propety is used to perform a very large file upload in chunks without exceeding the maximum request size configured for the web server. For example, the default file chunk size used by the IDE for file uploads is 16MB. The size and last modified properties are assigned to the file once it has been completely written to the target path. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: multipart/mixed; boundary=<Content Boundary>

Example Request:

```
https://localhost/administration?method=uploadfile
```

Example Request Content:

```
--85E52D200059154CADE514918E72C9
Content-Type: application/json; charset=utf-8

{
   "Name": "\/Content\/musiccollection.html",
   "Size": 173678,
   "LastModified": 1607220954332,
   "Offset": 0
}
--85E52D200059154CADE514918E72C9
Content-Type: text/html
Content-Disposition: attachment; filename="/Content/musiccollection.html"
```

```
<!doctype html>
<html>
<head>
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="user-scalable=no, width=device-width,
      initial-scale=1, minimum-scale=1, maximum-scale=1">
   <meta name="application-name" content="Music Collection Client
      Application">
   <meta name="application-version" content="1.00">
   <meta name="application-load-progress" content="False">
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
   <meta http-equiv="Content-Style-Type" content="text/css">
   <meta http-equiv="Content-Script-Type" content="text/javascript">
<body>
</body>
</html>
--85E52D200059154CADE514918E72C9--
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Upload Resource Request

The upload resource request is a multi-part, mixed request that uploads the virtual resource provided in the included JSON content as a .zip file. The .zip file is first unzipped and each file/folder is processed independently as a separate file upload in a similar manner to the upload file request. The offset propety is used to perform a very large file upload in chunks without exceeding the maximum request size configured for the web server. For example, the default file chunk size used by the IDE for file uploads is 16MB. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: multipart/mixed; boundary=<Content Boundary>

Example Request:

```
https://localhost/administration?method=uploadresource
```

Example Request Content:

```
--591265089C5596441D169D4B2D62A5
Content-Type: application/json; charset=utf-8'

{
   "Name": "Content",
   "Size": 67516340,
   "Offset": 0
}
--591265089C5596441D169D4B2D62A5
Content-Type: application/x-zip-compressed

<Binary ZIP File Content>
--591265089C5596441D169D4B2D62A5--
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Download Files Request

The download files request downloads the files provided in the included JSON content as a .zip file. The offset propety is used to perform a very large file upload in chunks without exceeding the maximum request size configured for the web server. For example, the default file chunk size used by the IDE for file uploads is 16MB. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: GET

HTTP Request Content Type: application/json; charset=utf-8
HTTP Response Content Type: application/x-zip-compressed

Example Request:

```
https://localhost/administration?method=downloadfiles
```

Example Request Content:

```
{
   "Files": ["\/Logs\/System\/20200725.elog",
             "\/Logs\/System\/20200727.elog",
             "\/Logs\/System\/20200724.elog",
             "\/Logs\/System\/20200723.elog",
             "\/Logs\/System\/20200722.elog",
             "\/Logs\/System\/20200721.elog"]
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Rename File Request

The remame file request renames the file provided in the included JSON content to a new name. The file name is a URL path that should begin with a virtual resource name and may contain sub-folders, but must refer to an existing valid file already present on the web server. The new file name should not contain any path information. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=renamefile
```

Example Request Content:

```
{
```

```
   "Name": "\/Content\/computer.png",
   "NewName": "pc.png"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Remove File Request

The remove file request removes the file provided in the included JSON content. The file name is a URL path that should begin with a virtual resource name and may contain sub-folders, but must refer to an existing valid file already present on the web server. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=removefile
```

Example Request Content:

```
{
   "Name": "\/Content\/computer.png"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Create Folder Request

The create folder request creates the folder provided in the included JSON content. The folder is a URL path that should begin with a virtual resource name and may contain sub-folders, but must refer to a valid path for the new folder. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=createfolder
```

Example Request Content:

```
{
```

```
   "Name": "\/Content\/Images"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Rename Folder Request

The remame folder request renames the folder provided in the included JSON content to a new name. The folder is a URL path that should begin with a virtual resource name and may contain sub-folders, but must refer to an existing valid path already present on the web server. The new folder name should not contain any path information. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=renamefolder
```

Example Request Content:

```
{
   "Name": "\/Content\/Images",
   "NewName": "Img"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Remove Folder Request

The remove folder request removes the folder provided in the included JSON content. The folder is a URL path that should begin with a virtual resource name and may contain sub-folders, but must refer to an existing valid path already present on the web server. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: POST

HTTP Request Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=removefolder
```

Example Request Content:

```
{
   "Name": "\/Content\/Images"
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

Get Files Request

The get applications request enumerates all of the files in the file specification provided in the included JSON content. The file specification should begin with a virtual resource name and may contain sub-folders, but must refer to an existing valid path already present on the web server. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: GET

HTTP Request Content Type: application/json; charset=utf-8
HTTP Response Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=getfiles
```

Example Request Content:

```
{
   "Name": "\/Applications\/*.*"
}
```

Example Response Content:

```
{
   "Name": "Applications",
   "Files": [{ "Name": "excepttest.wba",
               "Size": 62245,
               "LastModified": 1597457504313 },
             { "Name": "musiccollection.wba",
               "Size": 78450,
               "LastModified": 1603738472323 },
             { "Name": "NewServerTest",
               "Files": [{ "Name": "project1.wba",
                           "Size": 54571,
                           "LastModified": 1595372868000 }] },
             { "Name": "photoalbum.wba",
                "Size": 64533,
                "LastModified": 1599675169367 },
             { "Name": "project1.wba",
               "Size": 66356,
               "LastModified": 1594945532641 },
             { "Name": "ServerApp",
               "Files": [{ "Name": "serverapp.wba",
                           "Size": 70798,
                           "LastModified": 1595372868000 }] }]
}
```

HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

2.24 Web Server Administration API - Status

Get Status Request

The get status request enumerates all of the statistics on the web server. If there is no active session or the session is not authenticated, the request will result in a 403 Forbidden HTTP response.

HTTP Method: GET

HTTP Response Content Type: application/json; charset=utf-8

Example Request:

```
https://localhost/administration?method=getstatus
```

Example Response Content:

```
{
   "UpTime": 530,
   "MemoryUsage": 131769148,
   "NumSessions": 1,
   "HTTP": { "NumConnections": 1,
             "RequestsPerSecond": 0,
             "ReadsPerSecond": 0,
             "WritesPerSecond": 0 },
   "HTTPS": { "NumConnections": 0,
              "RequestsPerSecond": 0,
              "ReadsPerSecond": 0,
              "WritesPerSecond": 0 }
}
```

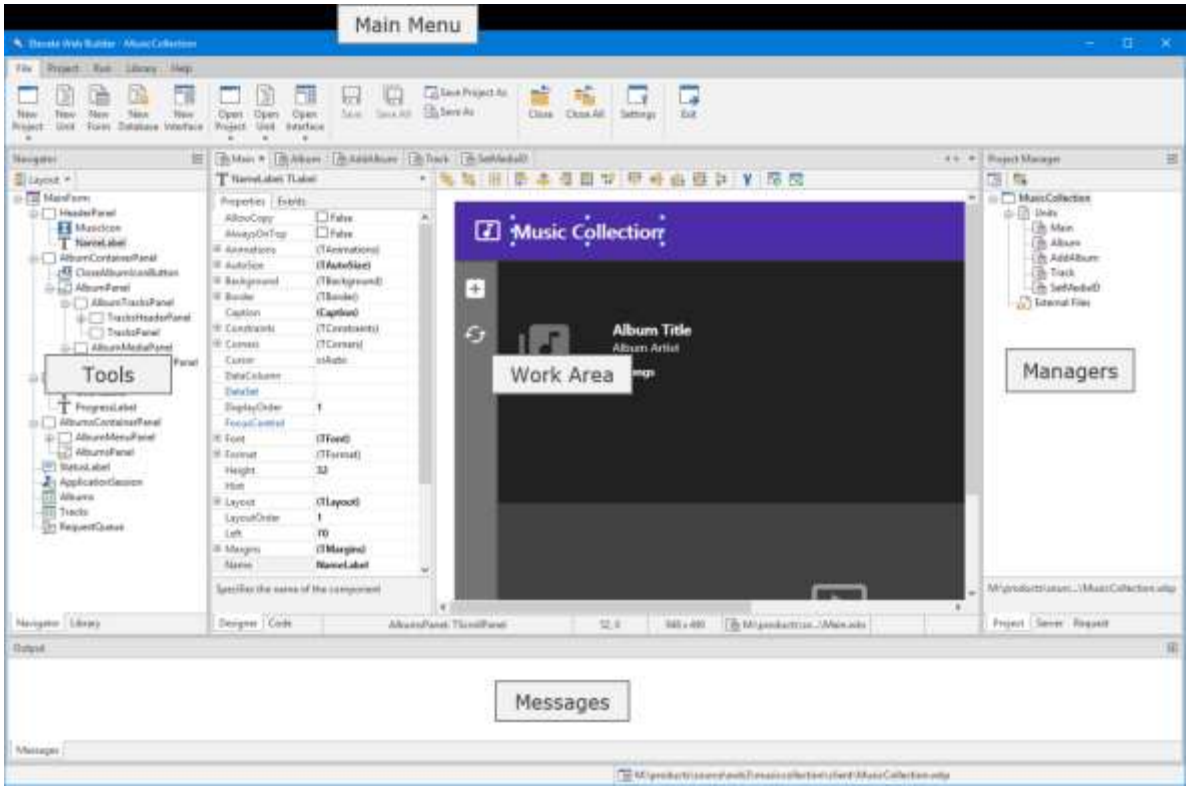
HTTP Response: 200 on success, 403 if not authenticated, and 500 on error

This page intentionally left blank

Chapter 3
Using the IDE

3.1 Introduction

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



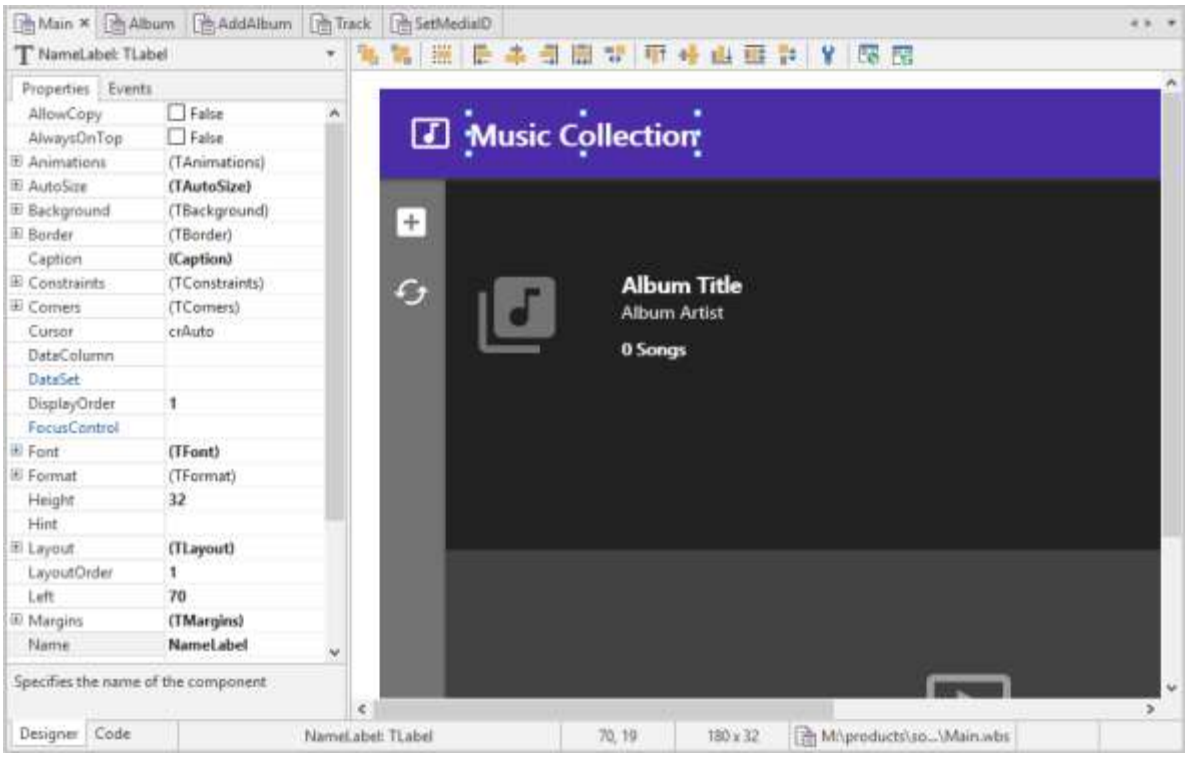
Main Menu

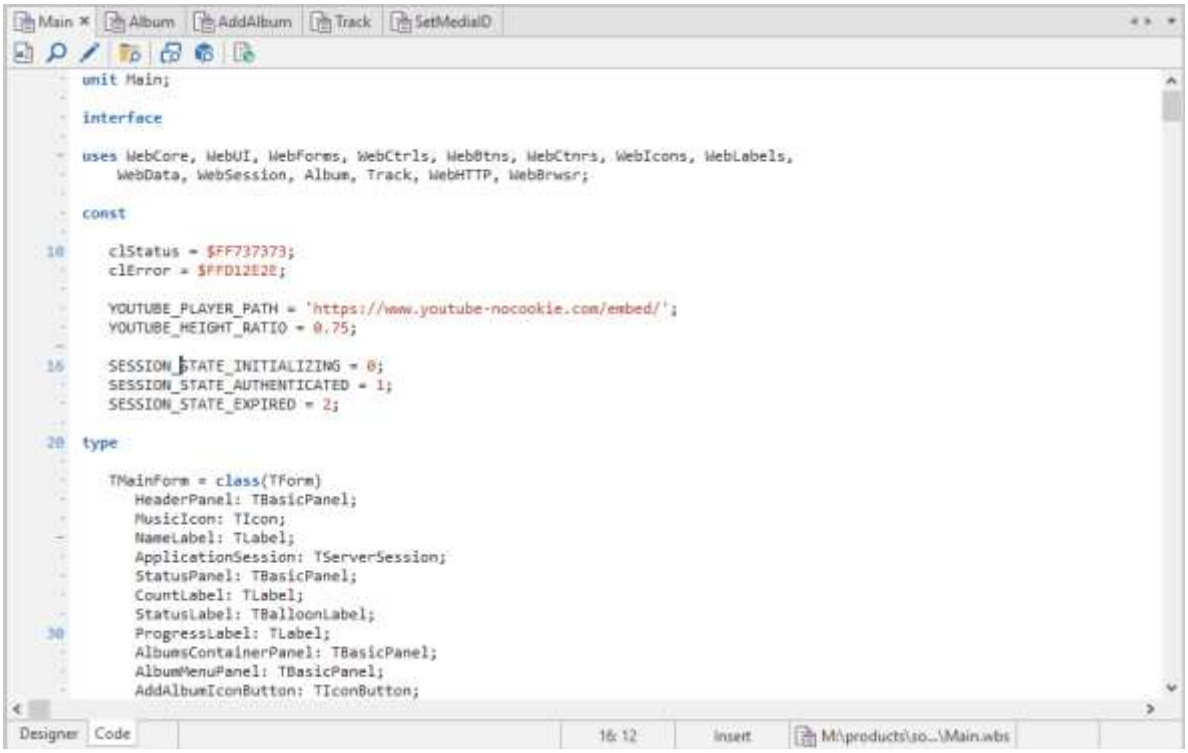
The main menu in the IDE provides options for:

- Creating new visual and non-visual client application projects and server application projects
- Creating new forms, databases, request handlers, and source units in a project
- Creating new control interfaces and modifying existing control interfaces
- Adding existing forms, databases, request handlers, and source units to an existing project
- Opening projects and source units
- Saving and closing projects and source units
- Viewing and editing IDE settings
- Viewing and editing project options
- Building and deploying projects

- Running and debugging projects
- Adding components to, removing components from, and rebuilding the component library
- Viewing and editing the icon library
- Generating interface units for native external modules (used with server applications)
- Accessing help and installing the example applications

Work Area

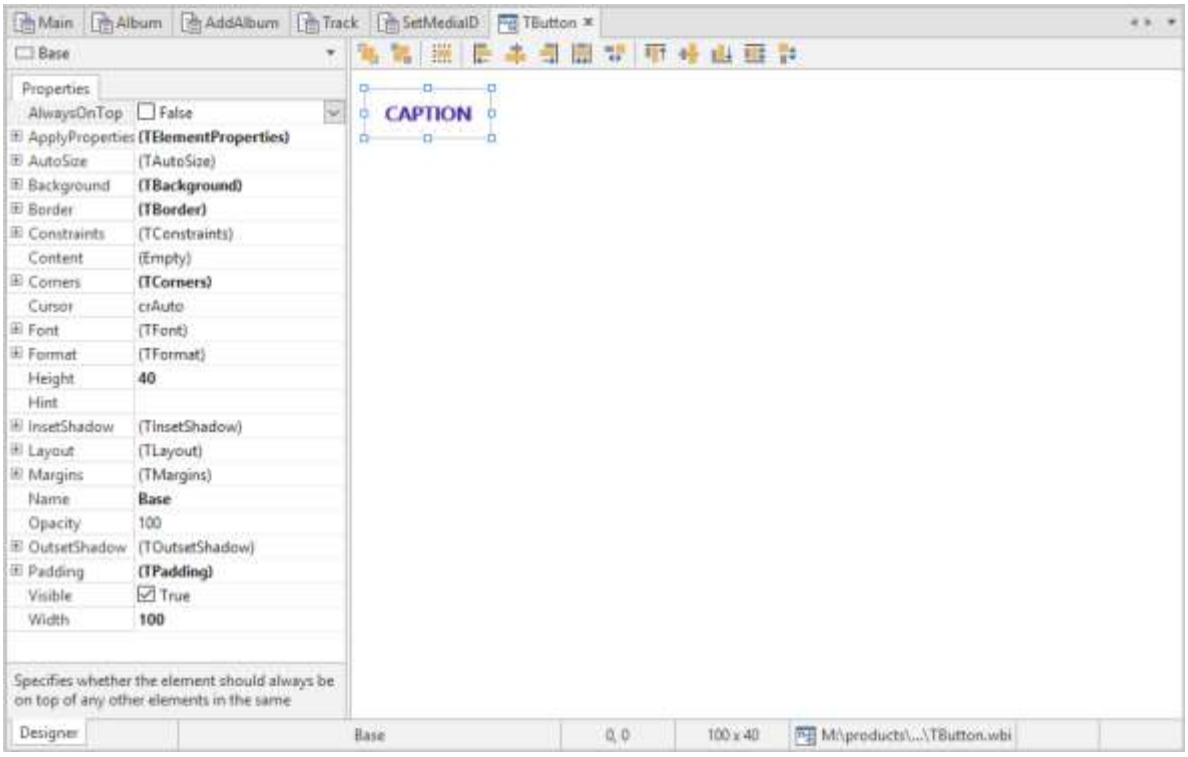




The work area is where all source units and, if applicable, their associated form, database, or request handler are docked when the source units are opened as part of a project or on their own. If a source unit has an associated form, database, or request handler, then both the component inspector and designer surface, as well as the code editor will be accessible. If a source unit does not have an associated form, database, or request handler, then only the code editor will be accessible. You can switch between the component inspector and designer surface and the code editor by using the F12 key or by clicking on the **Designer** or **Code** tab in the bottom left corner of the work area.

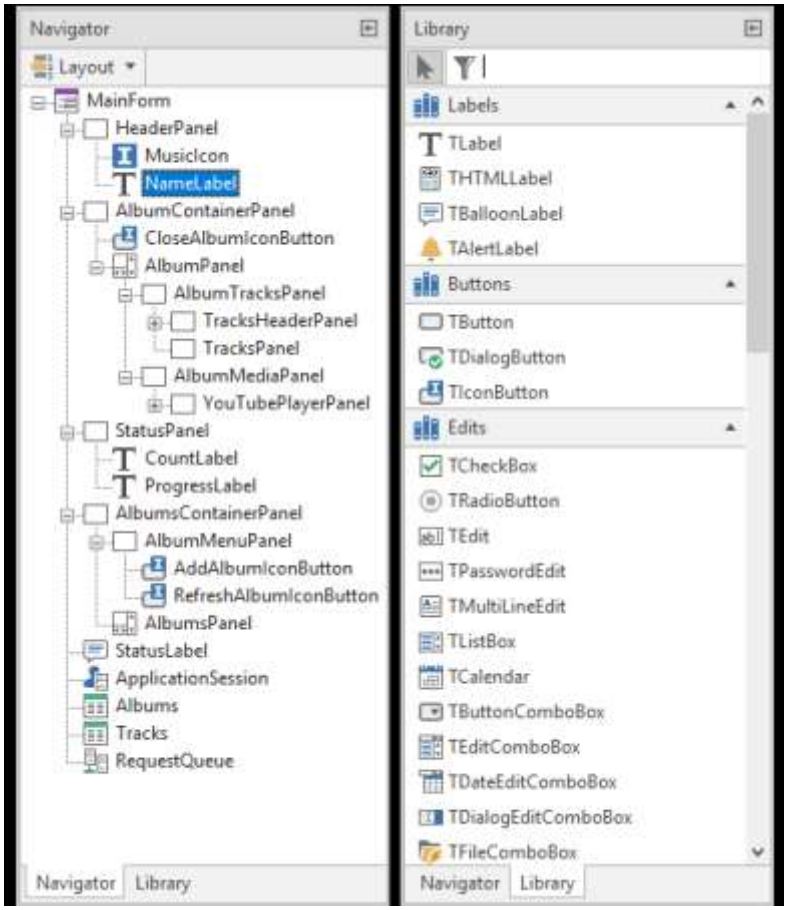
> **Note**
> The component inspector is always visible to the left of the designer surface when a form, database, or request handler source unit is open in the IDE and the designer surface is active.

In addition to source units, the work area is also where control interfaces are docked when they are opened. Control interfaces also use a component inspector and designer surface, but do not have a corresponding code editor. Control interfaces are not associated with a specific project, but open control interfaces are saved with open projects so that they are re-opened whenever the project is re-opened.



Tools

The tools area of the IDE is adaptive and will contain different tools depending upon the active state of the IDE. When a form, database, or request handler source unit is open and selected in the IDE and the designer surface is active, the tools area will contain two tools, the component navigator and the component library.



The component navigator allows you to select a component to work with, rename components, and change the layout or tab order of components. The component library allows you to search for and select a component to add to the designer surface for the active form, database, or request handler.

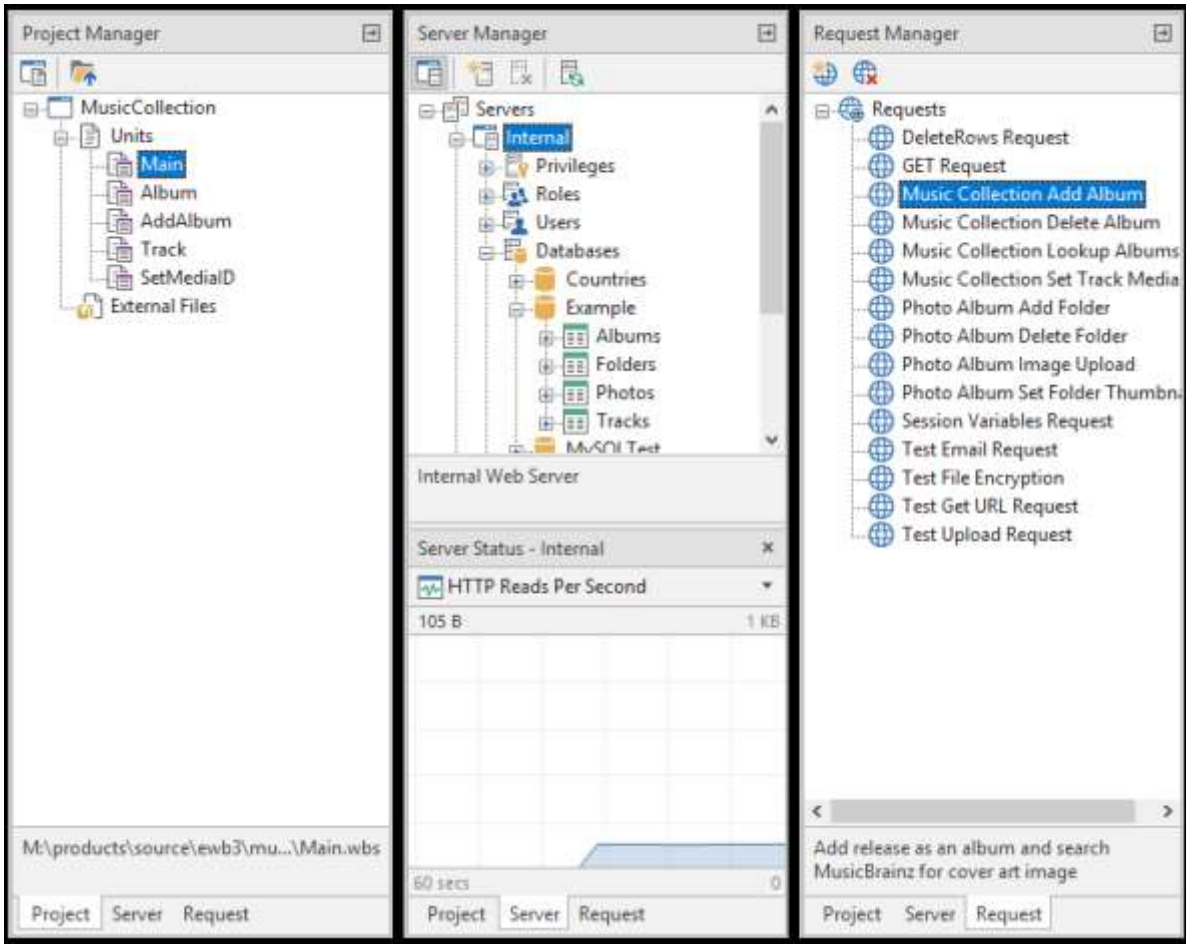
> **Note**
> There is both a client and server component library, so the components available in the component library will depend upon whether the open project is a client application project or a server application project.

When any source unit is open and selected in the IDE and the code editor is active, the tools area will be empty if the open project is a client application project, or will contain the source breakpoints if the open project is a server application project. The source breakpoints allow you to add and remove breakpoints in source units, as well as navigate to a given breakpoint.

When a control interface is open and selected in the IDE, the tools area will contain the control interface state and element navigators.


Navigator
TButton
Normal
Hot
Focused
Pushed
Disabled
Normal State
Base
Client
Icon
FontIcon
Caption
Navigator

Managers



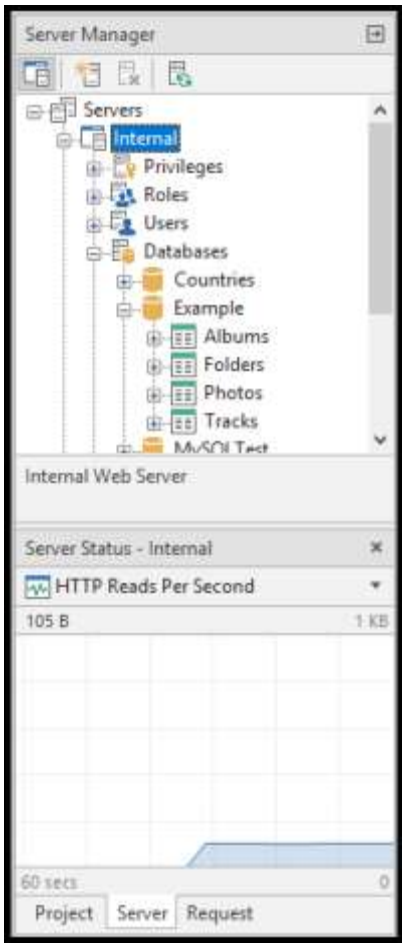
The managers area of the IDE always contains the project manager, the server manager, and the request manager. The project manager allows you to view the project source and add/remove both source units and external files to/from a project. You can also navigate to the containing folder for the project or any source units or external files. The server manager allows to connect to and remotely manage any Elevate Web Builder Web Server, including the Internal web server that is built into the IDE. The request manager allows you to add and remove web server requests that are used to execute and debug server applications in the IDE.

Messages

The messages area of the IDE is where all messages that are emitted from the IDE during the building, deployment, and debugging of applications are shown. In addition, the Internal web server in the IDE can emit additional messages such as debug output from client applications and and log output from web server migrations and file uploads and downloads.

3.2 Using the Server Manager

The server manager is used to define web servers that can be accessed by the IDE for the both client and server application deployment, remote server application debugging, and server administration. Please see the Deploying a Project topic for more information on how to select a web server when deploying a client or server application, the Running and Debugging a Project topic for more information on how to select a web server when debugging a server application, and the Web Server Administration topic for more information on using the server manager to administer a web server.



For ease of use, the IDE includes an embedded version of the Elevate Web Builder Web Server for development. By default, this web server is called **Internal** and is automatically set up during the IDE startup with a specific configuration for use with the **localhost** loopback host and the standard HTTP/HTTPS ports 80 and 443. You can use the instructions below on how to edit a web server in order to change any of the settings for the Internal web server. However, the Internal web server cannot be renamed or deleted in the server manager.

You can manually start and stop the Internal web server by clicking on the embedded web server button in the server manager toolbar:

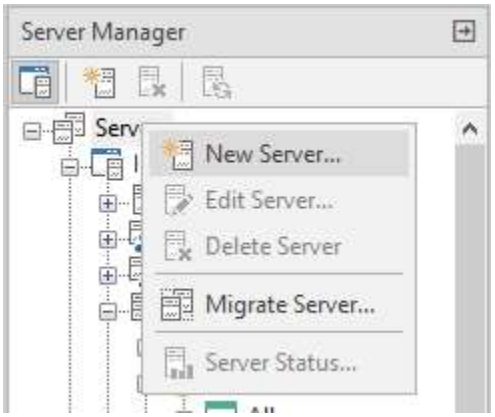
If the embedded web server button is selected, the Internal web server has been started.

Note
It is important to note that these web servers are being defined from the perspective of the IDE, and these settings have no bearing on how the web server instance being accessed by the IDE is configured. Therefore, these settings must be correct for the target web server instance or various aspects of the IDE, such as application deployment and server administration, will not work properly. The only exception to this is the Internal web server. For the Internal web server, the ports and resource names do actually control how the Internal web server is configured. Any changes to these settings for the Internal web server will result in the web server being automatically restarted in the IDE so that the new settings take effect.

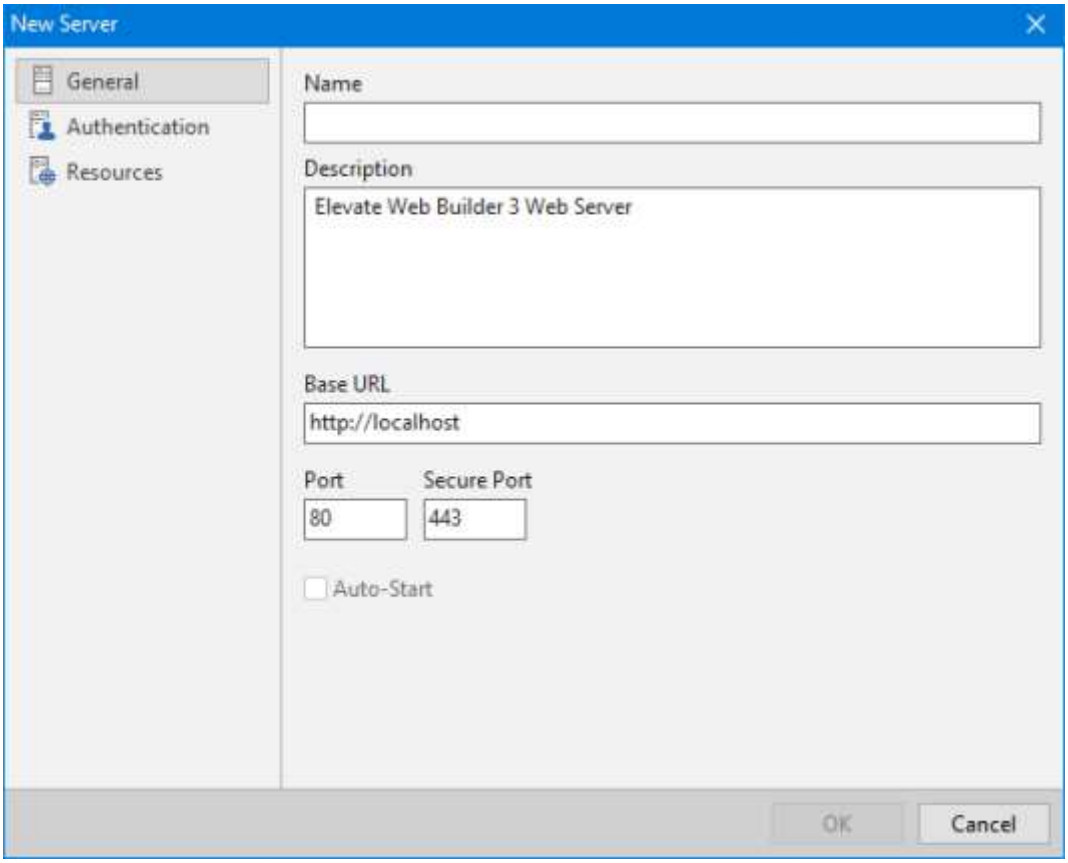
Creating a New Server

Use the following steps to create a new server using the server manager:

- Right-click on the **Servers** node of the server manager. The context menu for the server manager will appear. Click on the **New Server** menu option.



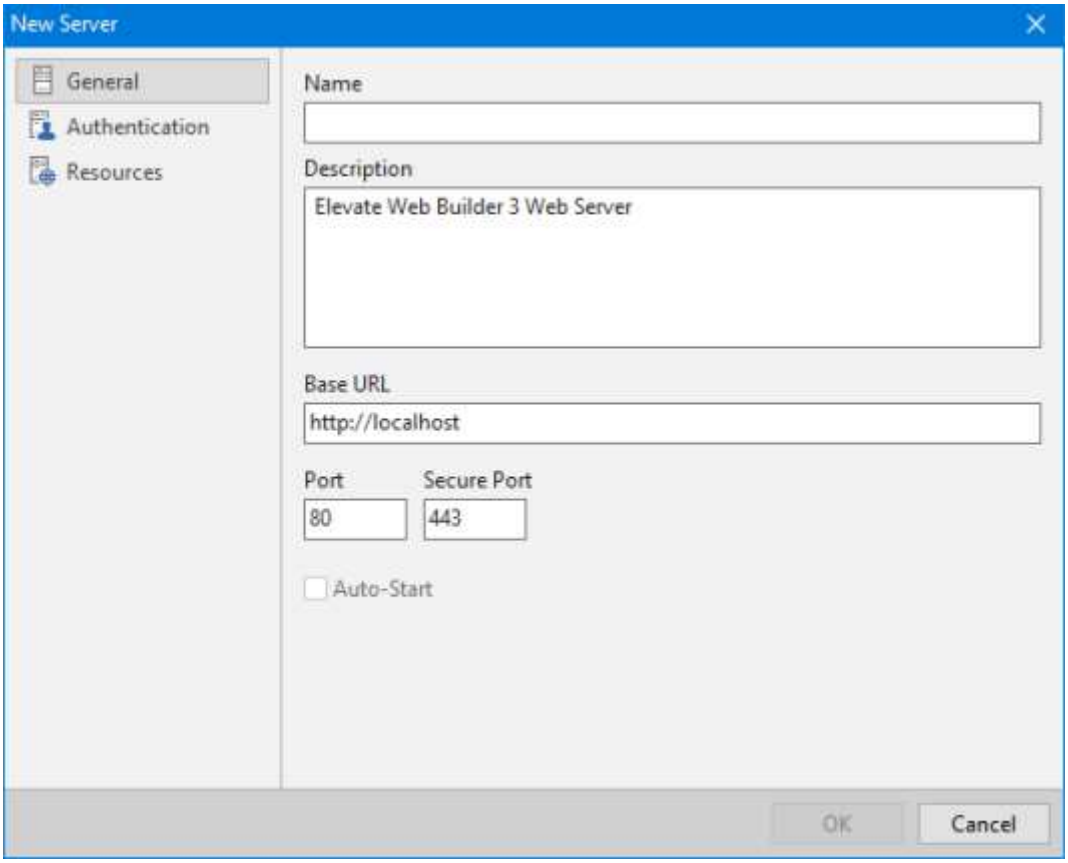
The New Server dialog will now appear:



After making any changes, click on the **OK** button to save the changes, or the **Cancel** button to discard the changes.

General

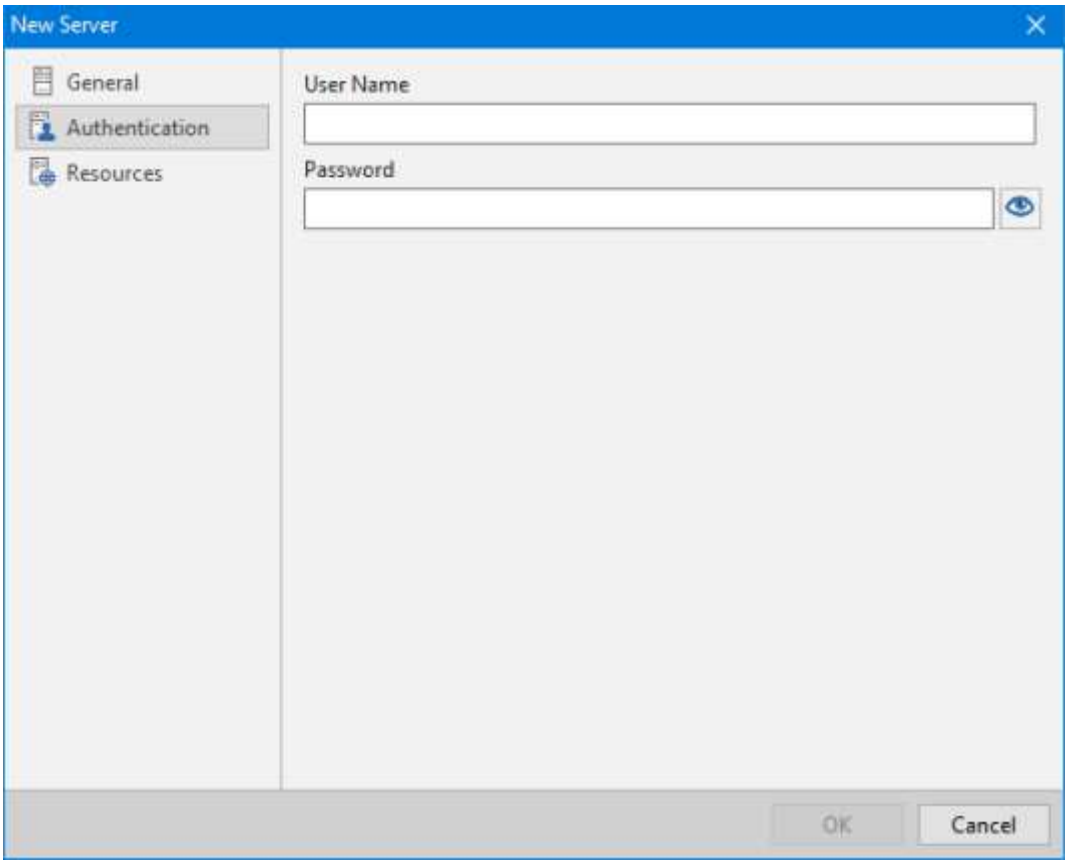
The General page provides options for specifying the name and description of the web server, along with the Base URL to use with any HTTP requests to the web server and the insecure HTTP and secure HTTPS ports to connect to when accessing the web server.



Setting	Description
Name	The name of the web server.
Description	The description of the web server.
Base URL	The base URL to use with any HTTP requests to the web server. The default base URL is "http://localhost". **Note** The base URL works in conjunction with the insecure HTTP port and secure HTTPS ports to determine whether access to the web server occurs in an insecure or secure manner. For example, if the base URL is "http://localhost", then any HTTP requests to the web server will be made to the insecure port. However, if the base URL is "https://localhost", then any HTTP requests to the web server will be made to the secure port.
Port	The port to use for insecure HTTP requests. The default port is 80.
Secure Port	The port to use for secure HTTP requests. The default secure port is 443.
Auto-Start	This check box is only enabled for the Internal web server and determines if the embedded web server is automatically started during the IDE startup. By default, the Internal web server is set to auto-start.

Authentication

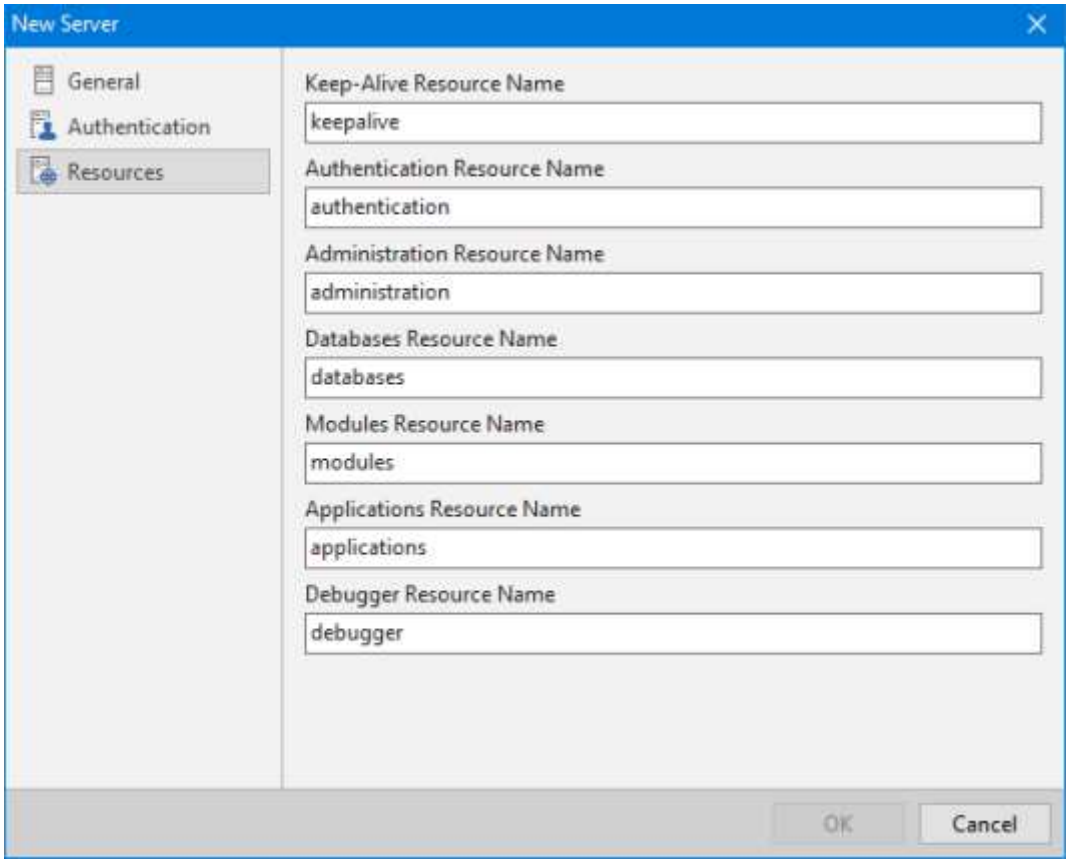
The Authentication page allows you to enter the user name and password to use during authentication. The IDE will first authenticate the user on the web server before making any HTTP requests, and will also authenticate the user on-demand if the web server indicates that the server session has expired.



Setting	Description
User Name	The user name to use for authentication on the web server.
Password	The password for the user name. Click on the button with the eye icon in order to show the entered password.

Resources

The Resources page provides options for specifying the defined resources on the web server. These resource names are used in conjunction with the base URL defined on the General page to construct HTTP requests for the various APIs surfaced by the web server.

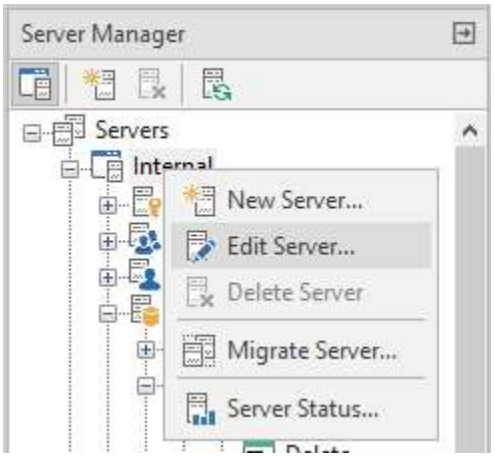


Setting	Description
Keep-Alive Resource Name	The resource name to use for keep-alive requests. The default value is "keepalive".
Authentication Resource Name	The resource name to use for authentication requests. The default value is "authentication".
Administration Resource Name	The resource name to use for administration requests. The default value is "administration".
Databases Resource Name	The resource name to use for database requests. The default value is "database".
Applications Resource Name	The resource name to use for server application requests. The default value is "applications".
Modules Resource Name	The resource name to use for native server module requests. The default value is "modules".
Debugger Resource Name	The resource name to use for debugger requests. The default value is "debugger".

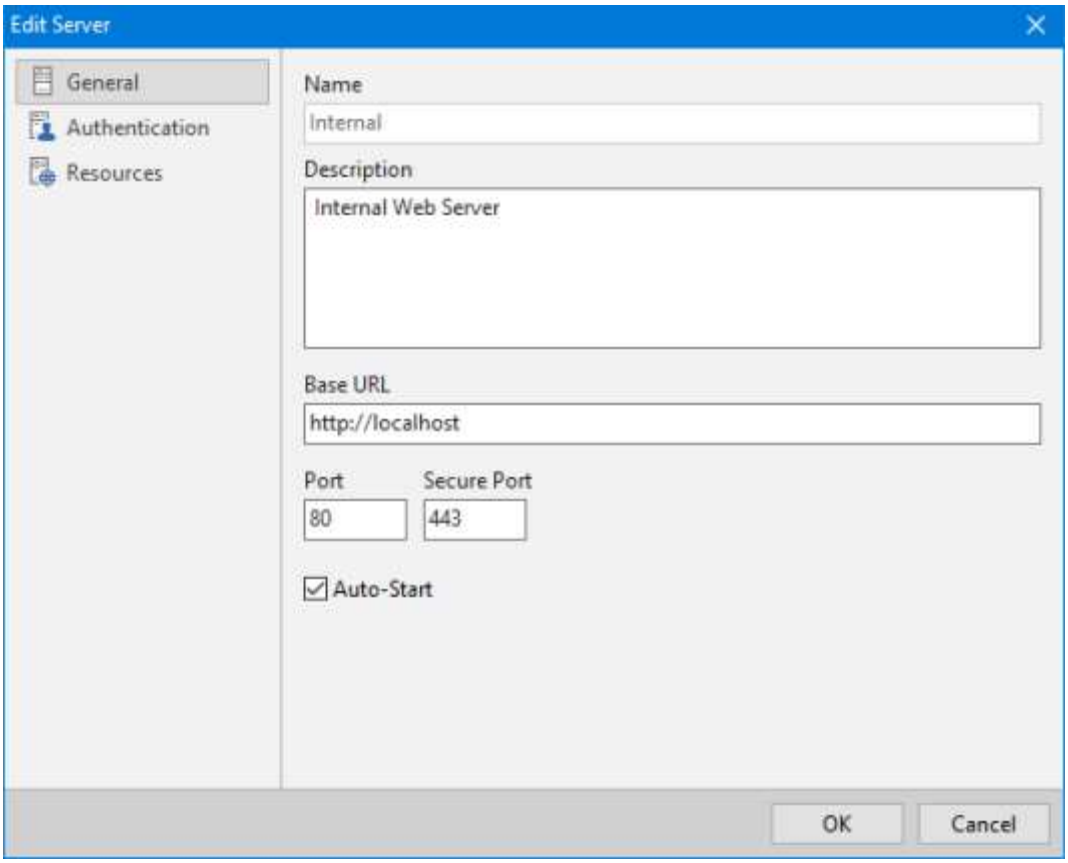
Editing an Existing Server

Use the following steps to edit an existing server using the server manager:

- Right-click on the server node that you want to edit. The context menu for the server manager will appear. Click on the **Edit Server** menu option.



The Edit Server dialog will now appear:

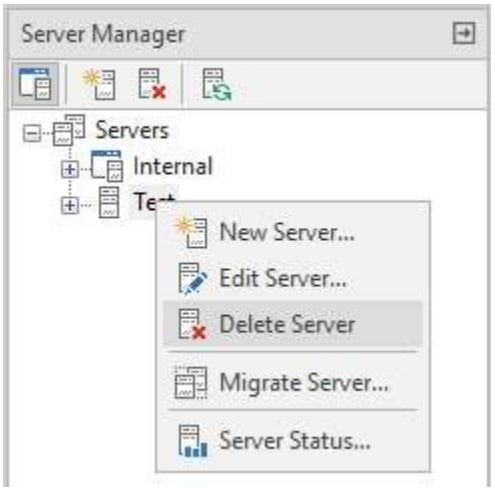


After making any changes, click on the **OK** button to save the changes, or the **Cancel** button to discard the changes.

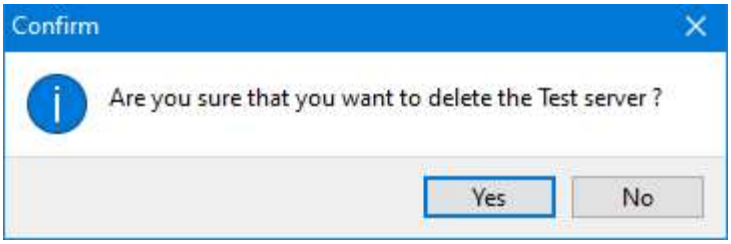
Deleting an Existing Server

Use the following steps to delete an existing server using the server manager:

- Right-click on the server node that you want to delete. The context menu for the server manager will appear. Click on the **Delete Server** menu option.



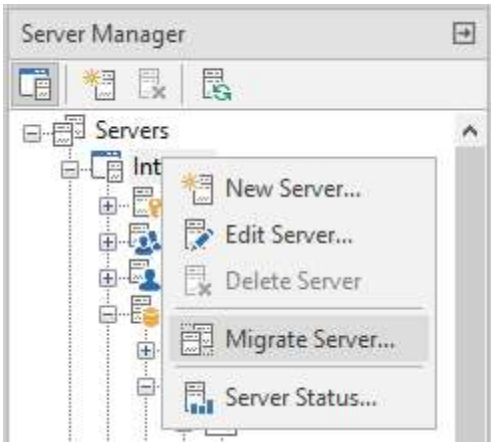
- A confirmation dialog will be displayed, asking you to confirm the removal of the server. Click on the **Yes** button to continue, or the **No** button to cancel the removal.



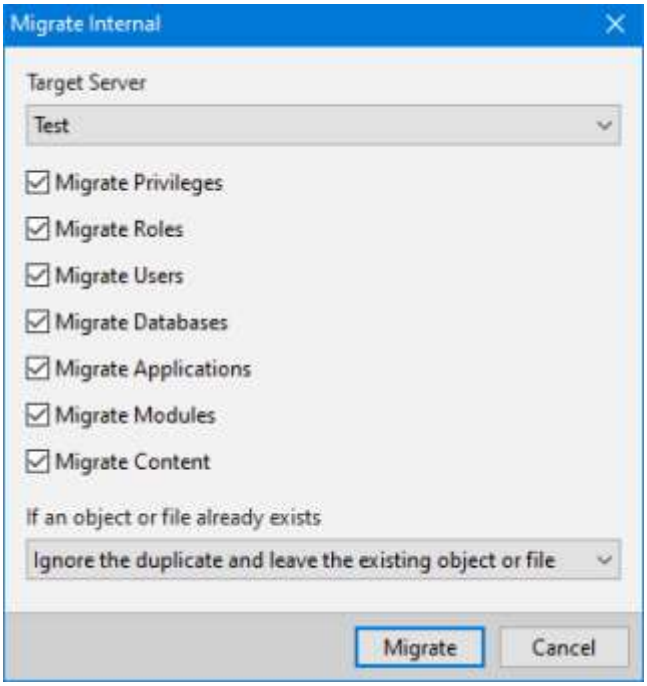
Migrating an Existing Server

You can easily migrate defined server objects and/or content from one existing server to another existing server. Use the following steps to execute a server migration using the server manager:

- Right-click on the server node that you want to migrate. The context menu for the server manager will appear. Click on the **Migrate Server** menu option.



The Migrate Server dialog will now appear:



After making any changes, click on the **Migrate** button to begin the migration, or the **Cancel** button to cancel the migration.

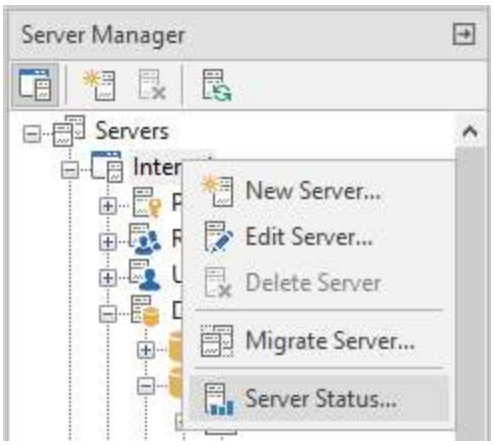
Option	Description
Target Server	Use this combo box to select the target web server for the migration.
Migrate Privileges	Use this check box to specify whether privileges should be migrated.
Migrate Roles	Use this check box to specify whether roles should be migrated.
Migrate Users	Use this check box to specify whether users should be migrated.
Migrate Databases	Use this check box to specify whether databases should be migrated.
Migrate Applications	Use this check box to specify whether installed server applications should be migrated. **Note** This option will cause all server applications to be migrated first, followed by the server applications being installed on the target web server.
Migrate Modules	Use this check box to specify whether installed native server modules should be migrated. **Note** This option will cause all native server modules to be migrated first, followed by the native server modules being installed on the target web server.
Migrate Content	Use this check box to specify whether all static content should be migrated.

Migration Conflicts	Use this combo box to select what to do if a server object or file that is being migrated already exists on the target web server.

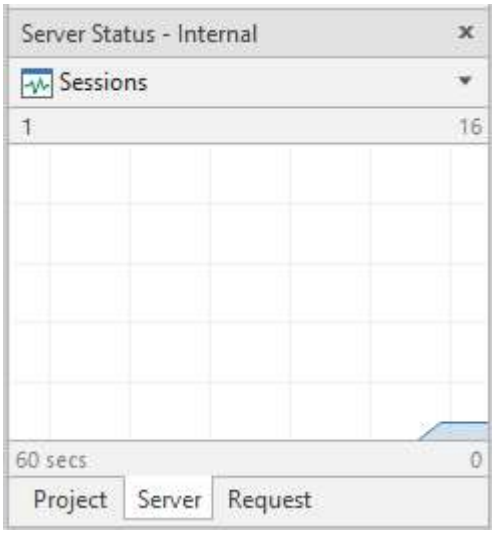
Viewing the Status of a Server

Use the following steps to view the status of a server using the server manager:

- Right-click on the server node that you want to examine. The context menu for the server manager will appear. Click on the **Server Status** menu option.



The server status panel will now appear at the bottom of the server manager:



You can use the combo box to select which statistic you would like to view. The server statistics are updated in real time every 3 seconds.

Note
Certain statistics for the embedded Internal web server are not valid because the web server is embedded in the IDE. For example, the Memory Usage statistic represents the total amount of memory used by the entire IDE, not just the Internal web server.

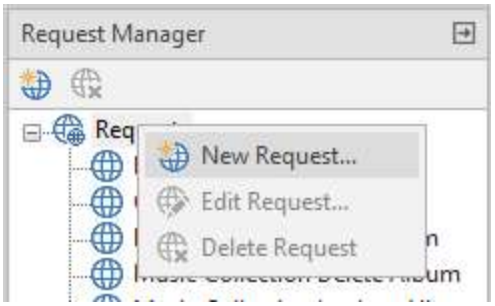
3.3 Using the Request Manager

The request manager is used to define HTTP requests that are used to remotely debug server applications in the IDE. Please see the Running and Debugging a Project topic for more information on how to select an HTTP request to use when debugging a server application, and the Server Request Architecture topic for more information about server requests.

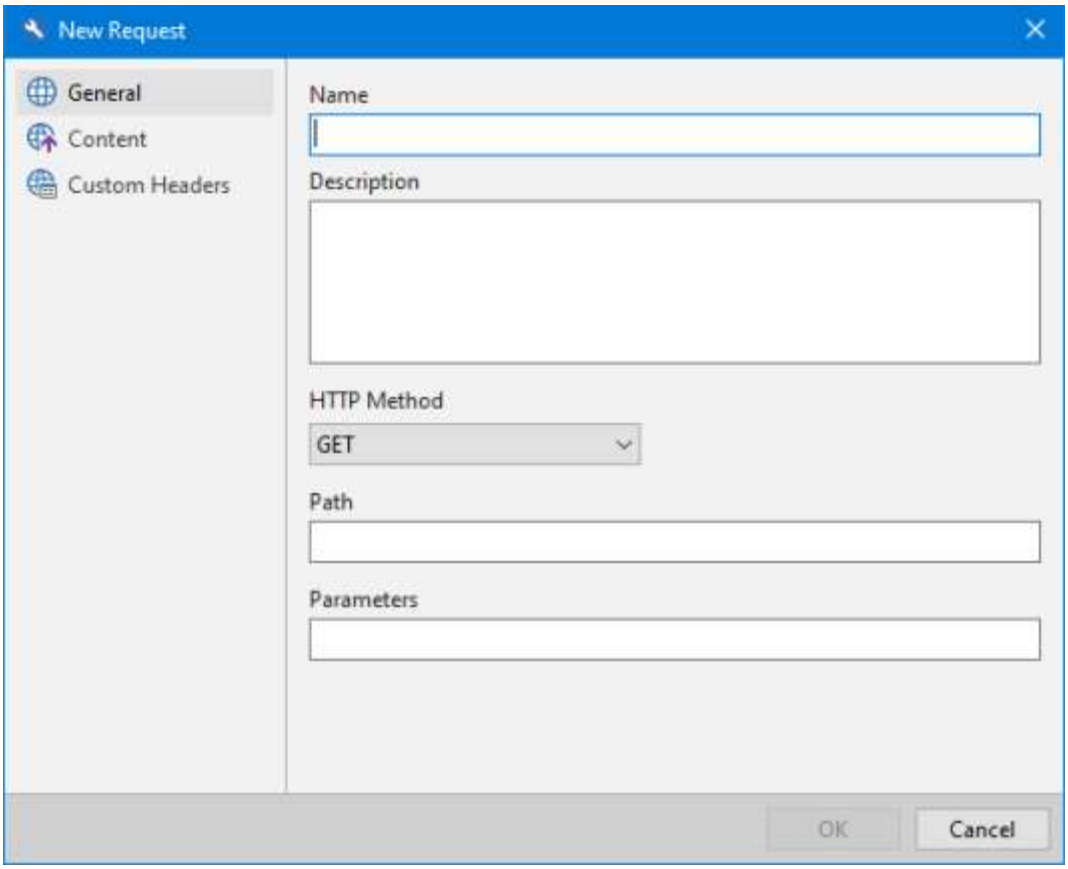
Creating a New Request

Use the following steps to create a new request using the request manager:

- Right-click on the **Requests** node of the request manager. The context menu for the request manager will appear. Click on the **New Request** menu option.



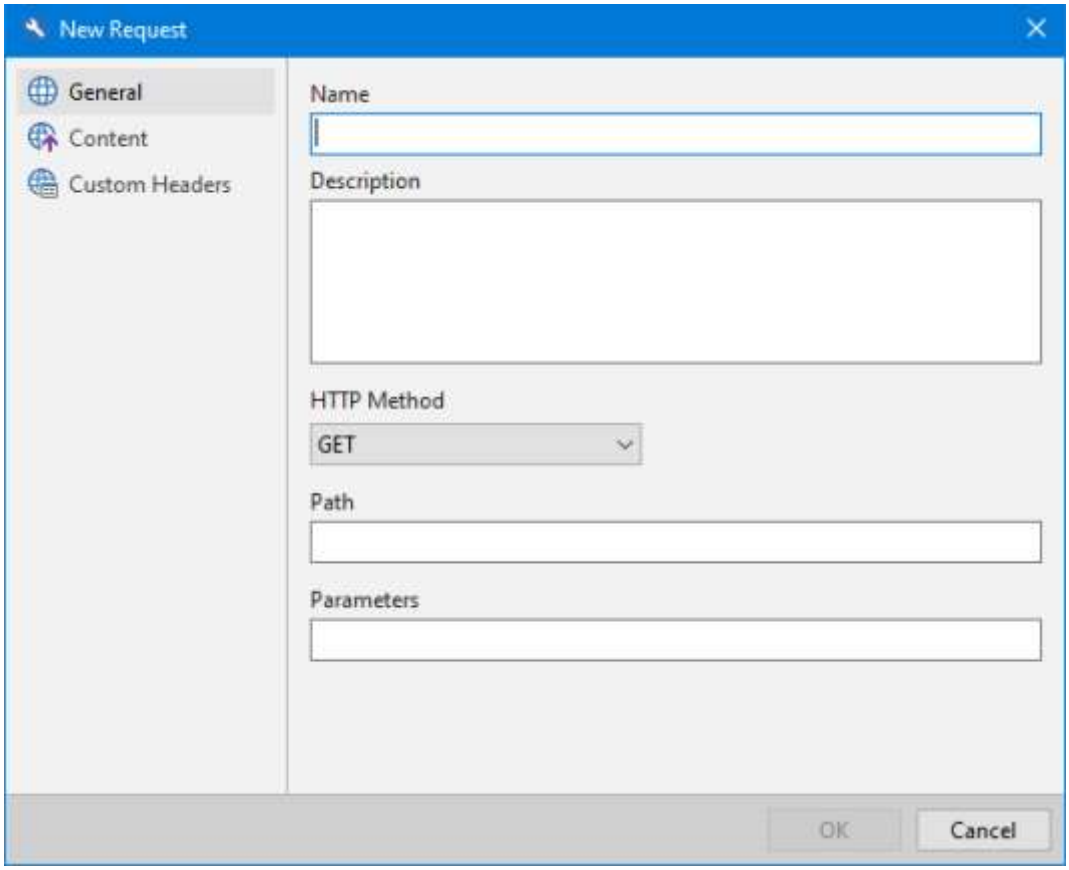
The New Request dialog will now appear:



After making any changes, click on the **OK** button to save the changes, or the **Cancel** button to discard the changes.

General

The General page provides options for specifying the name and description of the request, along with the HTTP method, relative URL path, and/or URL parameters to use with the request.



URL parameters are key-value pairs in the following form:

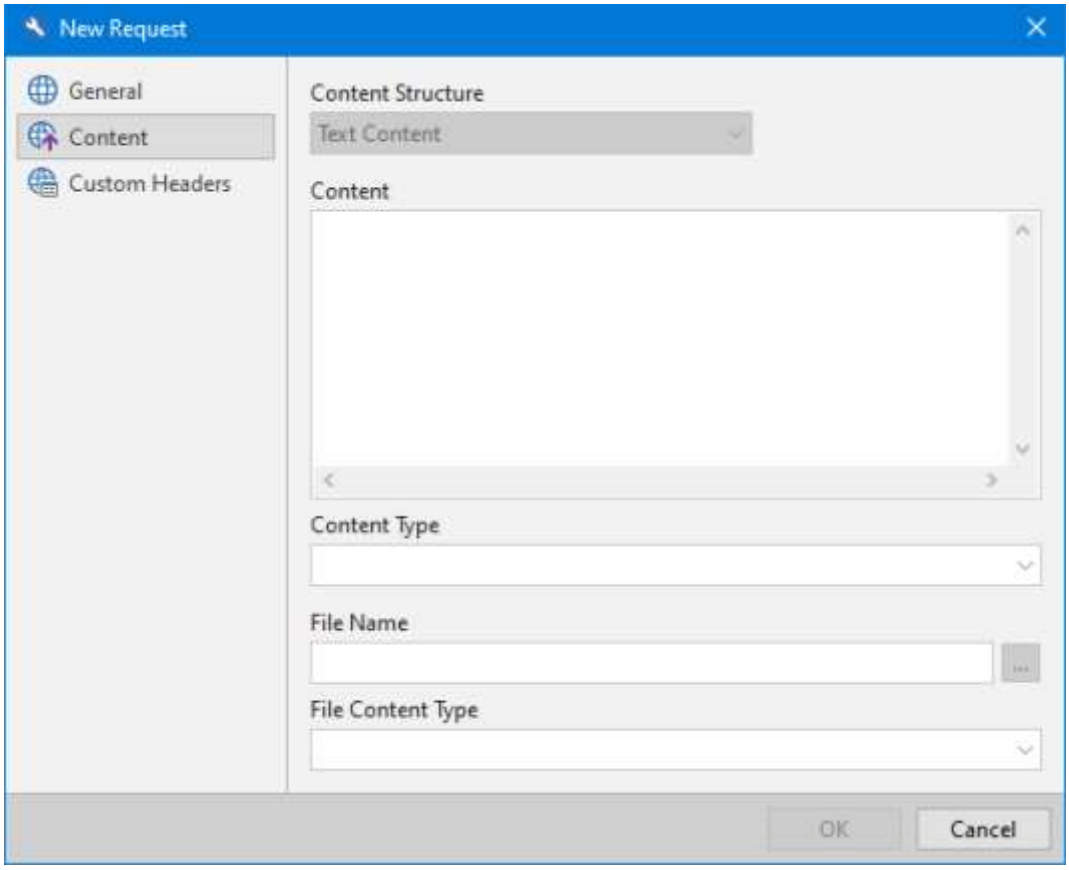
```
<Parameter Name>=<Parameter Value>[&<Parameter Name>=<Parameter Value>]
```

Setting	Description
Name	The name of the request.
Description	The description of the request.
HTTP Method	The HTTP method for the request. **Note** You can only specify content to be included with the request if the selected HTTP method is POST, PUT, or PATCH.
Path	This is a relative URL path that will be appended to the base URL path defined for the web server that the application is being debugged on. This path is useful with REST-ful HTTP API calls that use the path to identify a particular resource. **Note** Do not prefix this path with a path separator (/).
Parameters	These are the URL parameters that will be appended to the URL path derived from the base URL path defined for the web server that the application is being debugged on, combined with the

	relative URL path specified above. **Note** Do not prefix the parameters with a URL parameters separator (?). However, you do need to separate multiple parameters with the ampersand (&) parameter separator.

Content

The Content page provides options for defining the content to include with a POST, PUT, or PATCH request.

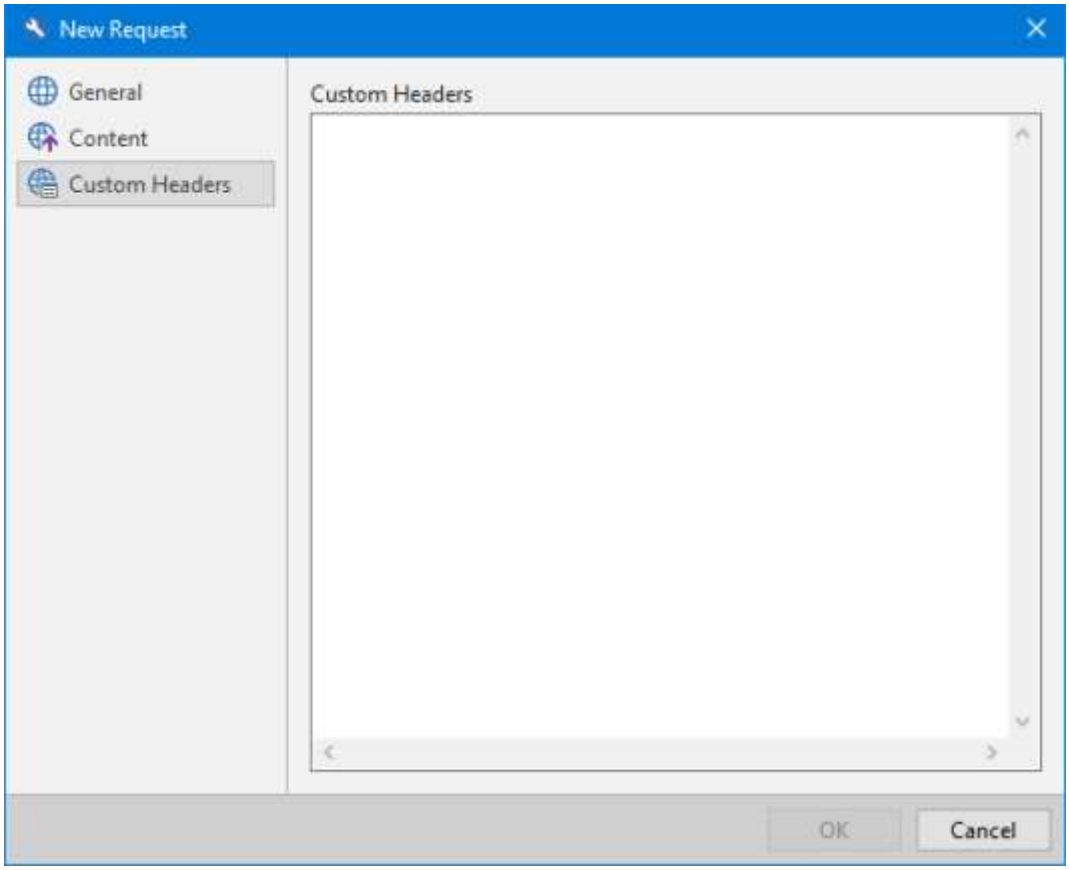


Setting	Description

Content Structure	Use this combo box to specify the structure of the content. The Text Content structure allows to include textual content that can be entered using the Content edit control, the File Content structure allows you to include a file as the content, and the Mixed Text/File Content and Mixed Form/File Content structures permit the inclusion of both textual content and file content. **Note** Any mixed content structure results in the content being encoded as multi-part content. For the Mixed Form/File Content structure, the Content Type cannot be specified because the textual content is automatically assumed to be URL-encoded, key-value data. This structure is synonymous with the multi-part form data requests that originate from submitted HTML forms.
Content	The textual content for the request.
Content Type	The MIME type of the textual content.
File Name	The file name to include as the file content for a mixed content structure.
File Content Type	The MIME type of the file content. **Note** If the file content type has not been populated, then it will be automatically populated based upon the file extension of the file name entered or selected.

Custom Headers

The Custom Headers page provides an area to add any custom headers that are necessary for the request.



Headers are key-value pairs in the following form:

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

By default, any HTTP requests are automatically set up with the following standard headers:

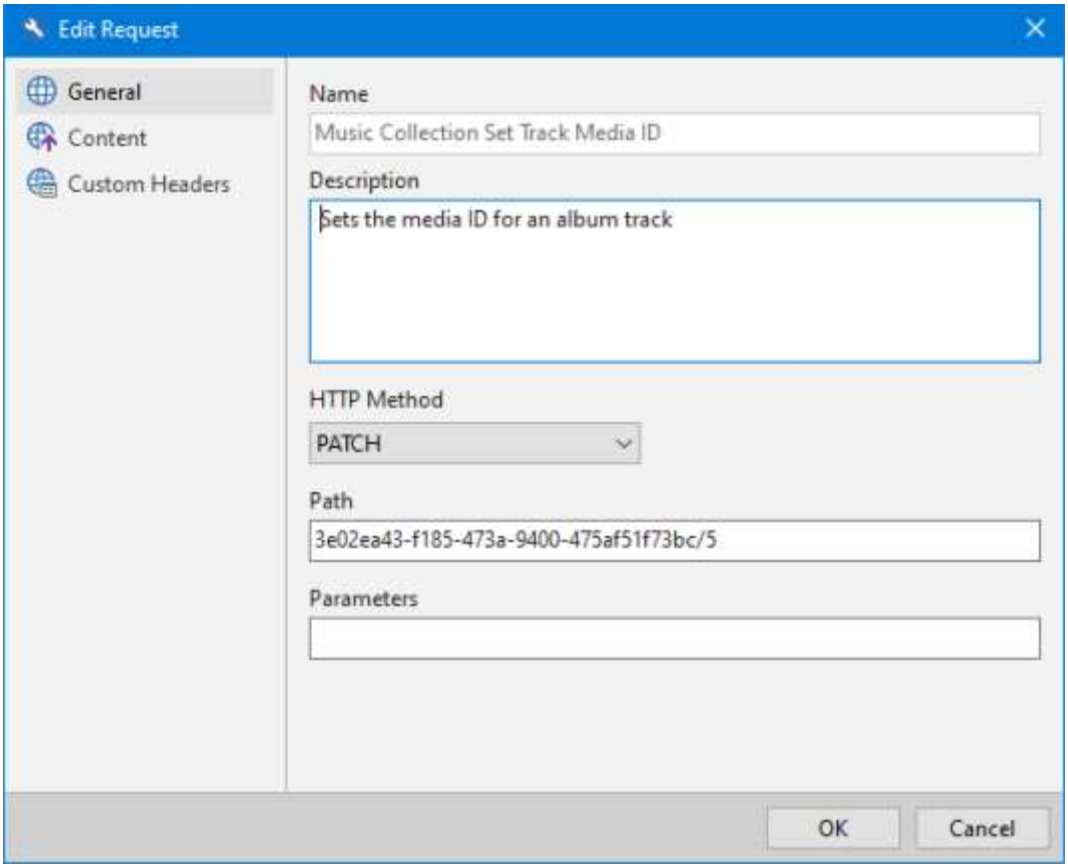
Header	Value
Date	Current date/time in RFC 7231 format <Day-Name>, <Day> <Month> <Year> <Hour>:<Minute>:<Second> GMT
User Agent	Populated programmatically, not used with requests defined in the request manager
Host	Host is derived from the URL for the target web server that the request is being executed against
Connection	Keep-Alive

Editing an Existing Request

Use the following steps to edit an existing request using the request manager:

- Right-click on the request node that you want to edit. The context menu for the request manager will appear. Click on the **Edit Request** menu option.

The Edit Request dialog will now appear:

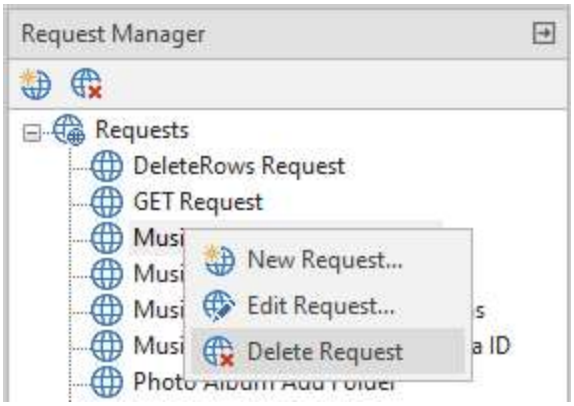


After making any changes, click on the **OK** button to save the changes, or the **Cancel** button to discard the changes.

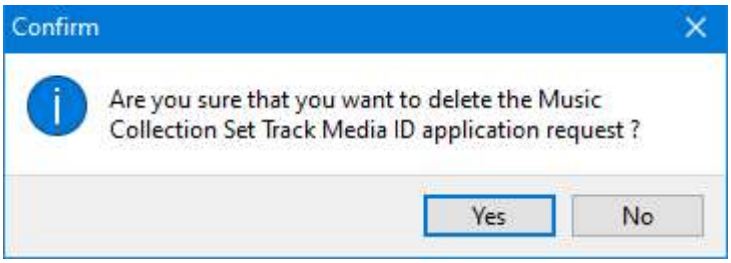
Deleting an Existing Request

Use the following steps to delete an existing request using the request manager:

- Right-click on the request node that you want to delete. The context menu for the request manager will appear. Click on the **Delete Request** menu option.



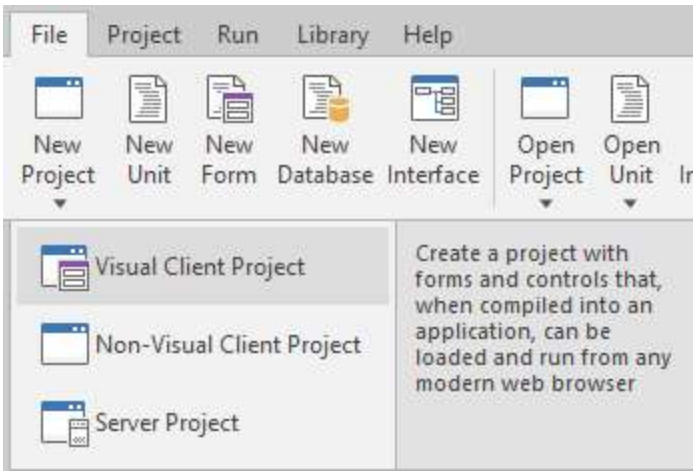
- A confirmation dialog will be displayed, asking you to confirm the removal of the request. Click on the **Yes** button to continue, or the **No** button to cancel the removal.



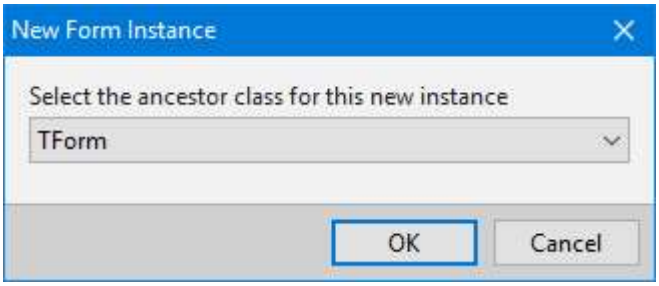
3.4 Creating a New Project

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

- Click on the **File** tab on the main menu.
- Click on the **New Project** button on the File menu to open the New Project sub-menu.

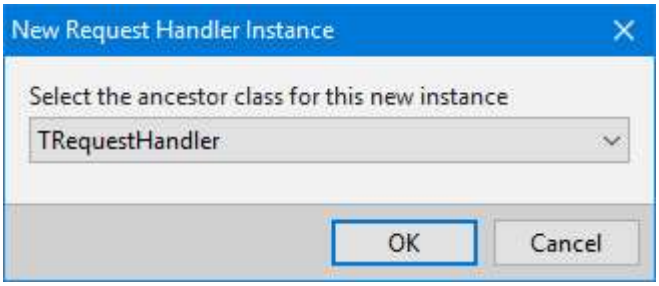


- Select the type of project that you would like to create: a visual client application, a non-visual client application, or a server application, and the new project will be opened in the IDE.
- If you selected a visual client application project type, you will then be prompted to select the base form class to use as the ancestor of the main form for the visual client application project:



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

- If you selected a server application project type, you will then be prompted to select the base request handler class to use as the ancestor of the main request handler for the server application project:



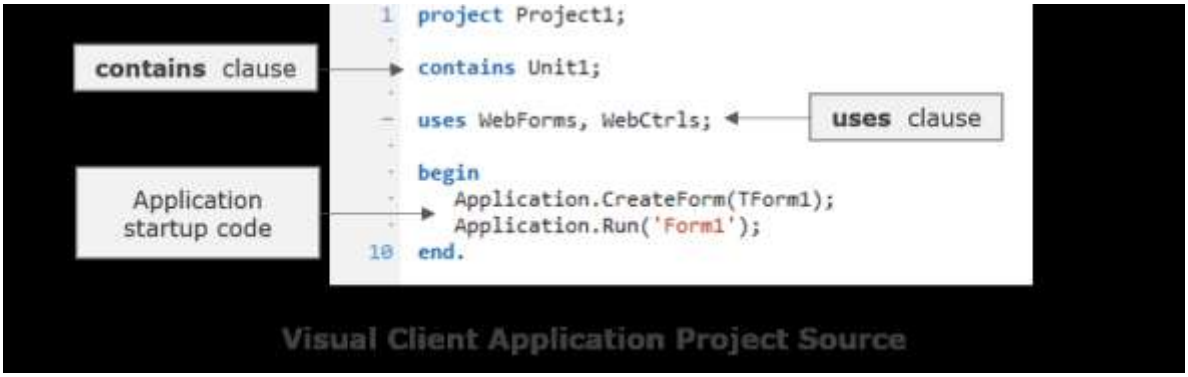
Currently, the component library only includes the **TRequestHandler** base request handler class.

- Click on the **OK** button.

Viewing the Project Source File

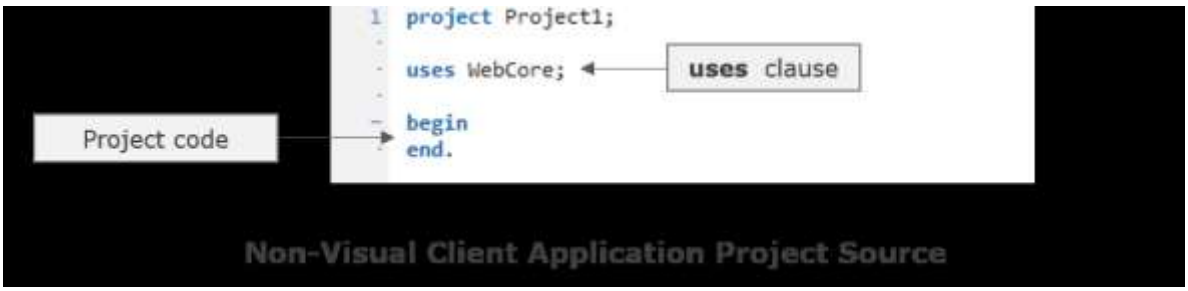
After creating the new project, you can open the project source file in the code editor by clicking the following button on the **Project Manager** toolbar:

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



Note
You'll notice that the IDE automatically inserts the **WebForms** and **WebCtrls** units into the **uses** clause. These units are necessary and should never be removed.

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

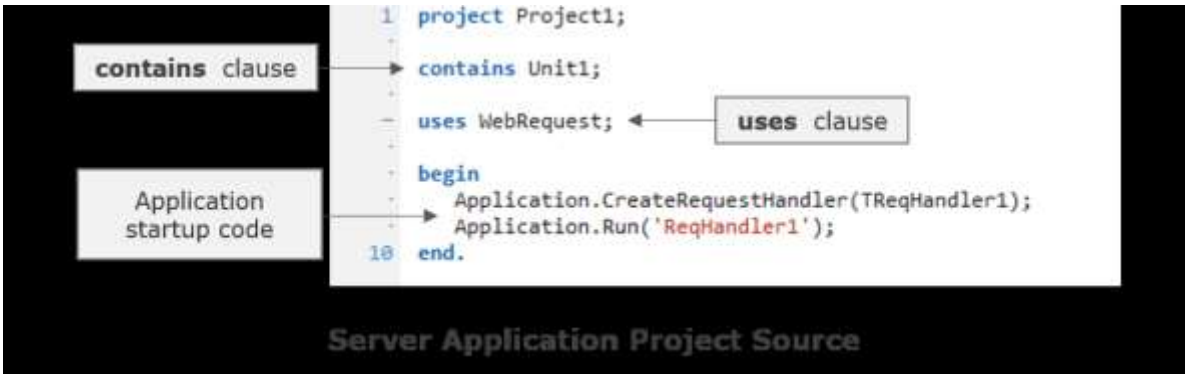


The user code would be added in the **begin..end** block.

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

For server application projects, the project source file's **contains** clause is updated to include the name of the source units included in the project, as well as the application startup code for the TApplication component that is used with server applications. In addition, any request handlers or databases that are designated as auto-create in

the Project Options are automatically created here. You can see how this looks in the following image:

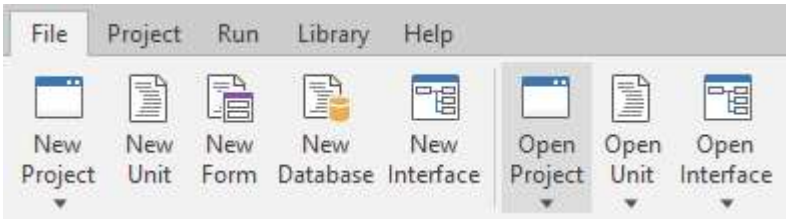


Note
You'll notice that the IDE automatically inserts the **WebRequest** unit into the **uses** clause. This unit is necessary and should never be removed.

3.5 Opening an Existing Project

Use the following steps to open an existing application project in the IDE:

- Click on the **File** tab on the main menu.
- Click on the **Open Project** button on the File menu.



A drop-down list of recently-opened projects will appear beneath the Open Project button. Select one of the recently-opened projects by clicking on the desired project, or select another project by clicking on the **Open Project from Folder** button at the bottom of the list of recently-opened projects.

> **Note**
> The IDE uses the presence of certain units in the **contains** section of the project source in order to determine what type of project is being opened. Please see the Application Structure topic for more information on how this works.

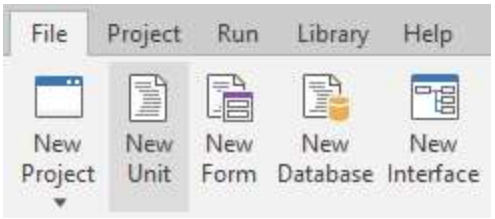
3.6 Adding to an Existing Project

You can easily add new source units, forms, request handlers, and databases to an existing project.

Adding a Source Unit to a Project (Visual and Non-Visual Client and Server)

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

- Click on the **File** tab on the main menu.
- Click on the **New Unit** button on the File menu.

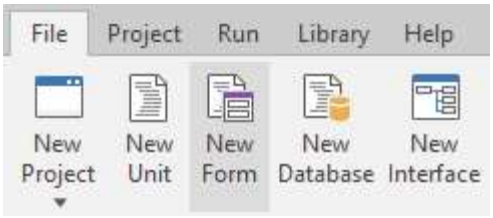


A new source unit will now appear in the work area of the IDE. Please see Using the Code Editor for more information on how to use the code editor.

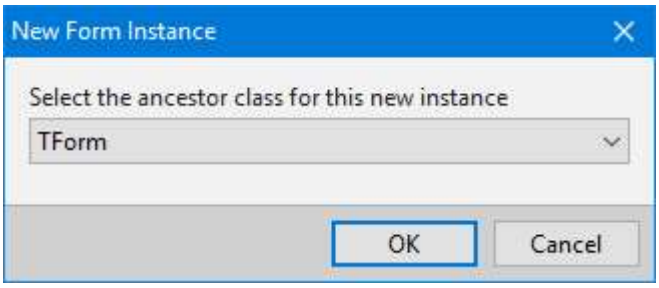
Adding a New Form to a Project (Visual Client)

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

- Click on the **File** tab on the main menu.
- Click on the **New Form** button on the File menu.



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



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

- Click on the **OK** button.

A new form will now appear in the work area of the IDE. Please see Using the Designer for more information on how

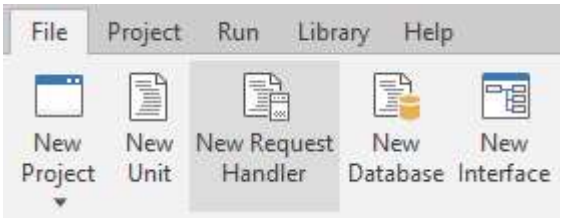
to use the form designer, and Using the Code Editor for more information on how to use the code editor.

> **Note**
> The New Form button is only available from the File menu when a visual client application project is active. You cannot add new forms to non-visual client application or server application projects.

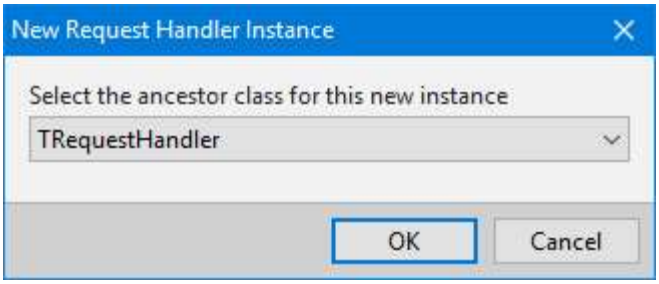
Adding a Request Handler to a Project (Server)

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

- Click on the **File** tab on the main menu.
- Click on the **New Request Handler** button on the File menu.



- You will then be prompted to select the type of request handler class to use as the ancestor of the request handler:



  If you're unsure as to which request handler class to use, just use the default **TRequestHandler** class.

- Click on the **OK** button.

A new request handler will now appear in the work area of the IDE. Please see Using the Designer for more information on how to use the request handler designer, and Using the Code Editor for more information on how to use the code editor.

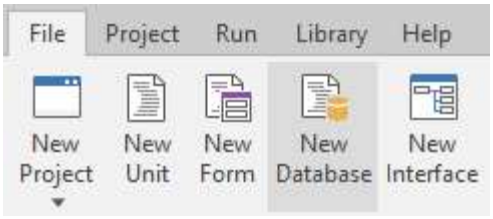
> **Note**
> The New Request Handler button is only available from the File menu when a server application project is active. You cannot add new request handlers to visual and non-visual client application projects.

Adding a Database to a Project (Visual Client and Server)

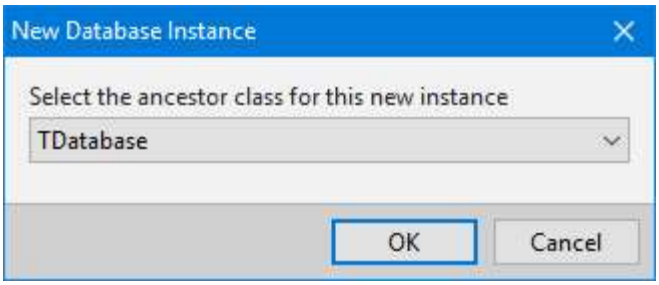
Use the following steps to add a new database to an existing project:

- Click on the **File** tab on the main menu.

- Click on the **New Database** button on the File menu.



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



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

- Click on the **OK** button.

A new database will now appear in the work area of the IDE. Please see Using the Designer for more information on how to use the database designer, and Using the Code Editor for more information on how to use the code editor.

> **Note**
> The New Database button is only available from the File menu when a visual client application or server application project is active. You cannot add new databases to non-visual client application projects.

3.7 Modifying Project Options

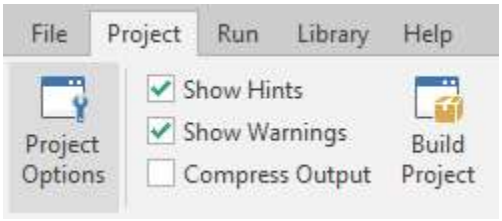
The project options for a project include:

- General application options (title, version number, icon, and whether to show load progress)
- Startup options (auto-created designer instances)
- Build options (hints/warnings, compiler defines, search paths, output paths and file names, output compression, and icon font options)

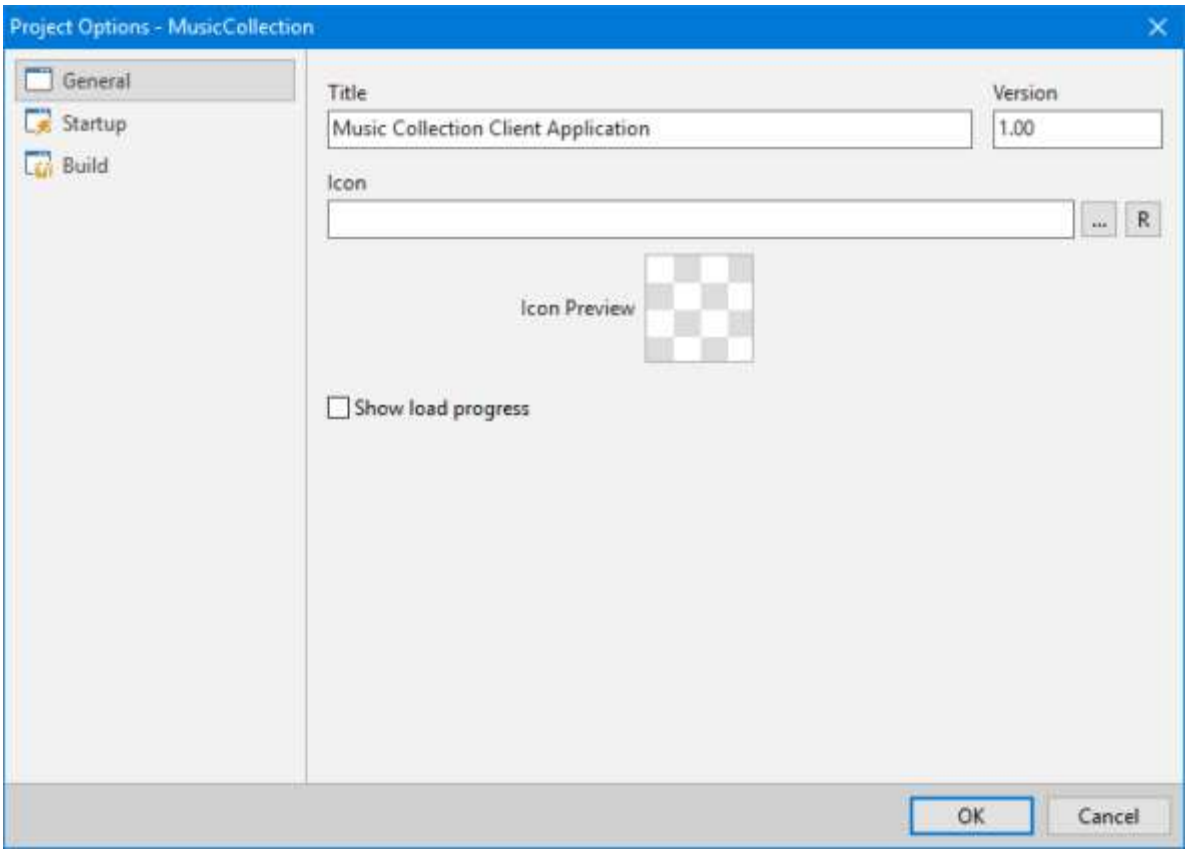
The available project options will differ depending upon whether the application is a visual client application, a non-visual client application, or a server application.

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

- Click on the **Project** tab on the main menu.
- Click on the **Project Options** button on the Project menu.



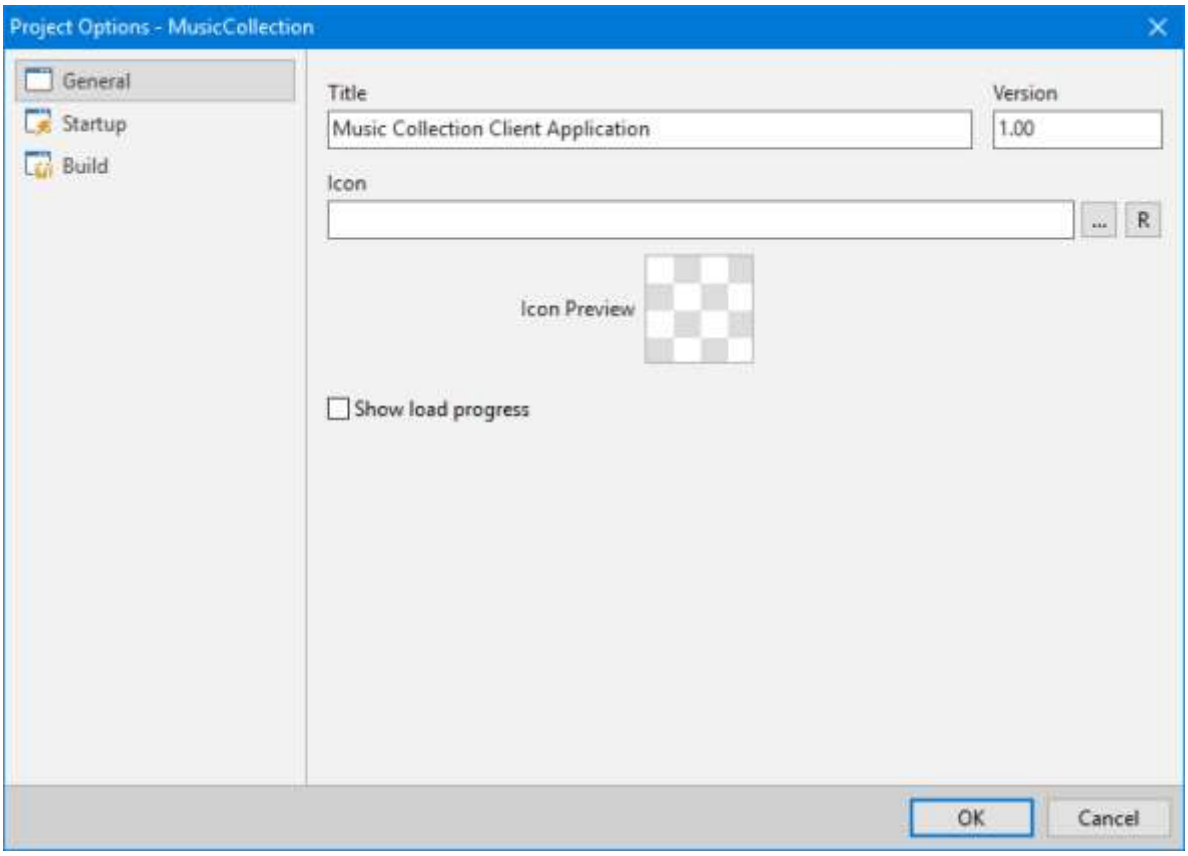
The Project Options dialog will now appear:



After making any changes, click on the **OK** button to save the changes, or the **Cancel** button to discard the changes.

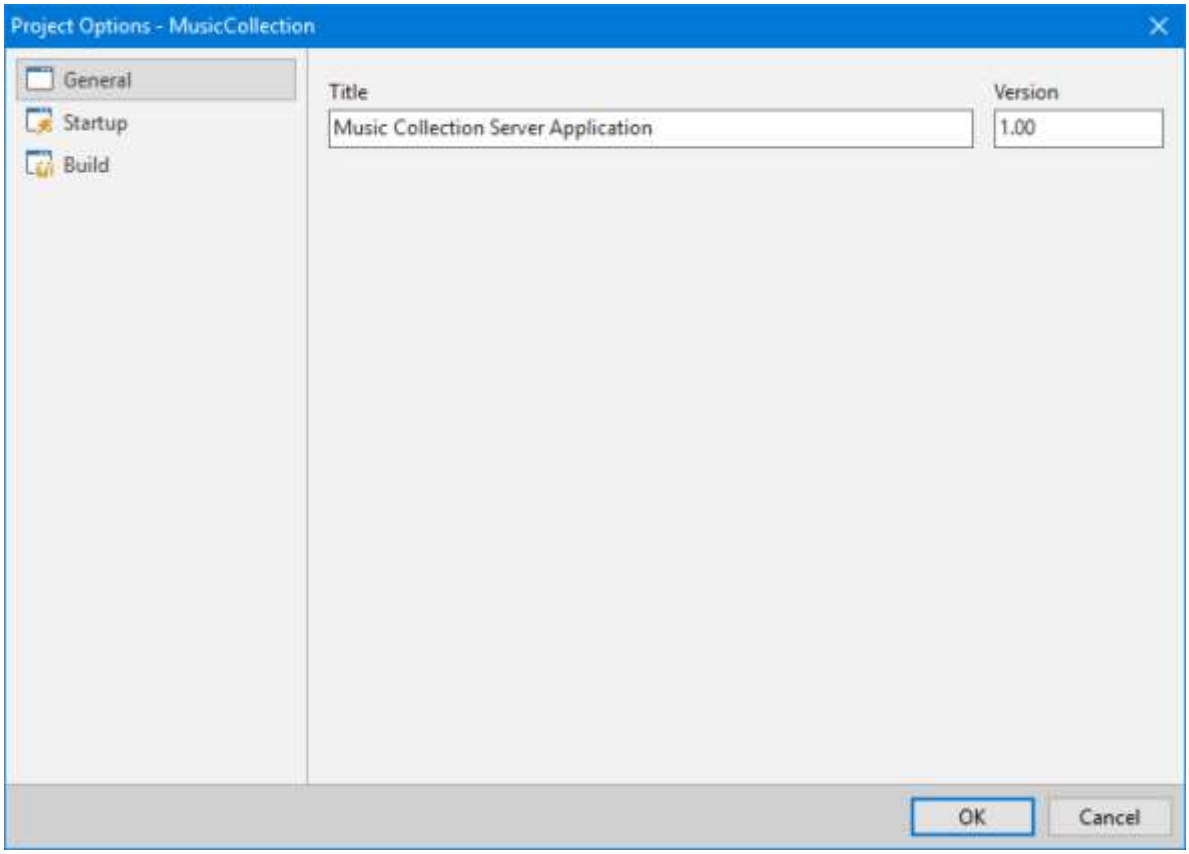
General Options (Visual Client and Server)

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



For non-visual client applications, the General page is not available.

For server applications the General page provides options for specifying the title and version number for the application.

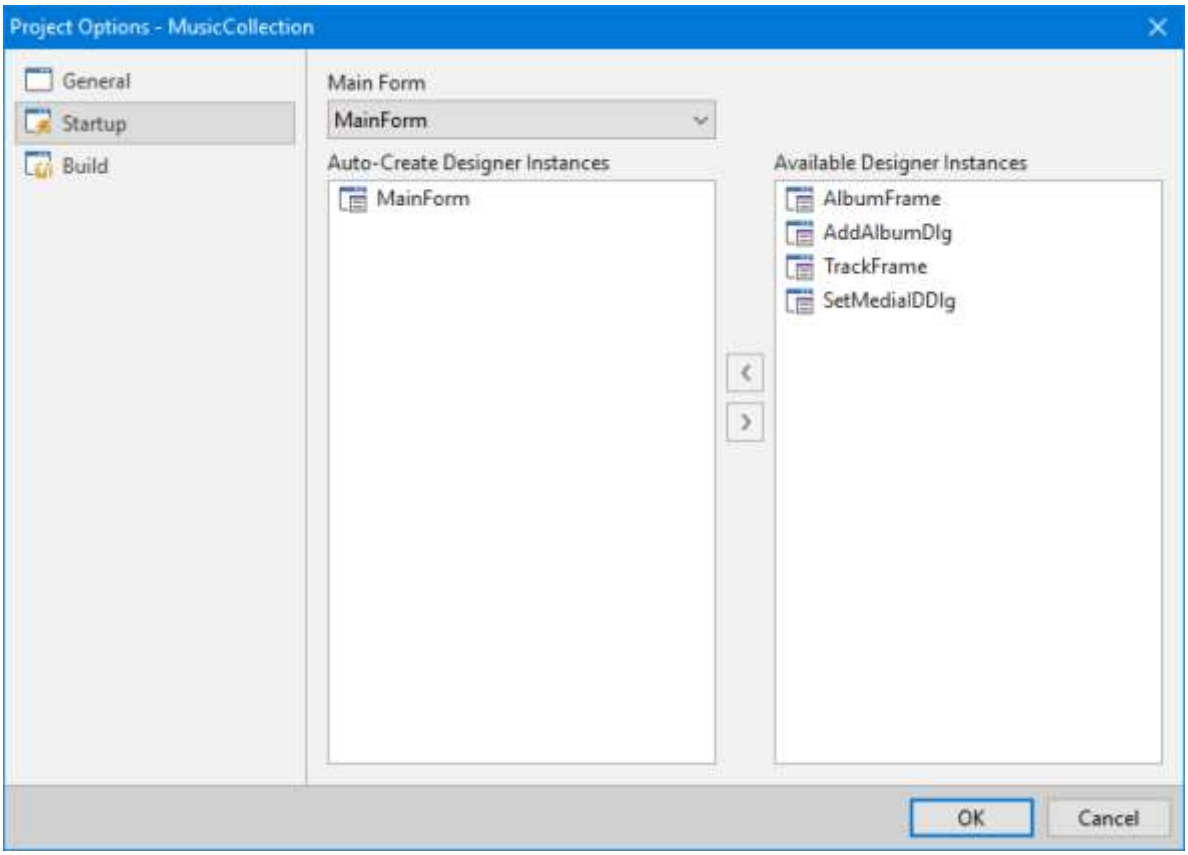


Option	Description
Title	The application title is the descriptive name for the application and, in most modern browsers, will appear in the caption bar of the browser window. This title is accessible to the application source code via the TApplication Title property.
Version	The application version is the version number to assign to the application. This version number is accessible to the application source code via the TApplication Version property.
Icon	The application icon is a 16x16 or 32x32 Windows icon file that is displayed in the browser window next to the application title. This icon is commonly known as a "favicon" (short for "favorite icon") because the icon is also used to help identify the application in "favorites" or "bookmarks" in the browser. You can type in the file name directly, or use the browse button (...) to select the icon file using a common Windows file dialog. After a valid file name has been specified or selected, a preview of the icon file will be shown in the Preview area. **Note** You do not have to specify an icon for an application and it is completely optional. Also, when selecting a file using the common Windows file selection dialog, the file name selected will contain an absolute path to the file. Click on the relative path (R) button to convert the file path to a path that is relative to the root project path.

Show load progress	If checked, this option will turn on the load progress dialog for the application. This dialog is shown during application startup. This setting is accessible to the application source code via the TApplication LoadProgress property.

Startup (Visual Client and Server)

For visual client applications, the Startup page allows you to specify which forms or databases in the project should be auto-created at application startup, as well as which form should be considered the main form for the application. The main form is automatically created and shown at application startup. The first form in the list of auto-create forms and databases is automatically designated as the main form of the application, but you can select a different form as the main form using the combo box at the top of the page.

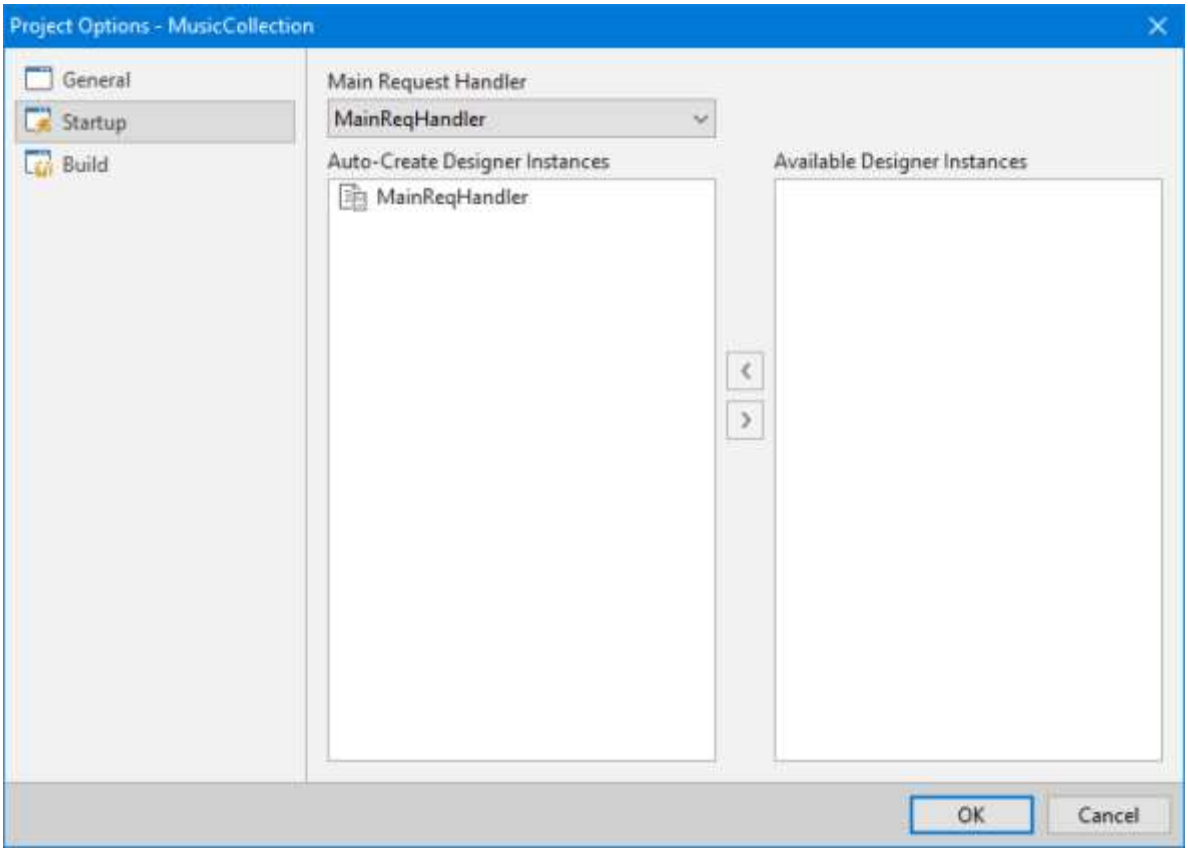


All updates to the main form and/or the auto-create forms and databases list will be reflected in the project source file. The following shows the project source file that corresponds to the auto-create forms and databases list above:

```
1  project MusicCollection;
-
-  contains Main, Album, AddAlbum, Track, SetMediaID;
-
-  uses WebForms, WebCtrls;
-
-  begin
-     Application.CreateForm(TMainForm);
-     Application.Run('MainForm');
10 end.
```

For non-visual client applications, the Startup page is not available.

For server applications, the Startup page allows you to specify which request handlers or databases in the project should be auto-created at application startup, as well as which request handler should be considered the main form for the application. The main request handler is automatically created and used as the routing point for the incoming web server request at application startup. The first request handler in the list of auto-create request handlers and databases is automatically designated as the main request handler of the application, but you can select a different request handler as the main request handler using the combo box at the top of the page.



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

```
1  project MusicCollection;
·
·  contains Main;
·
–  uses WebRequest;
·
·  begin
·     Application.CreateRequestHandler(TMainReqHandler);
·     Application.Run('MainReqHandler');
10 end.
```

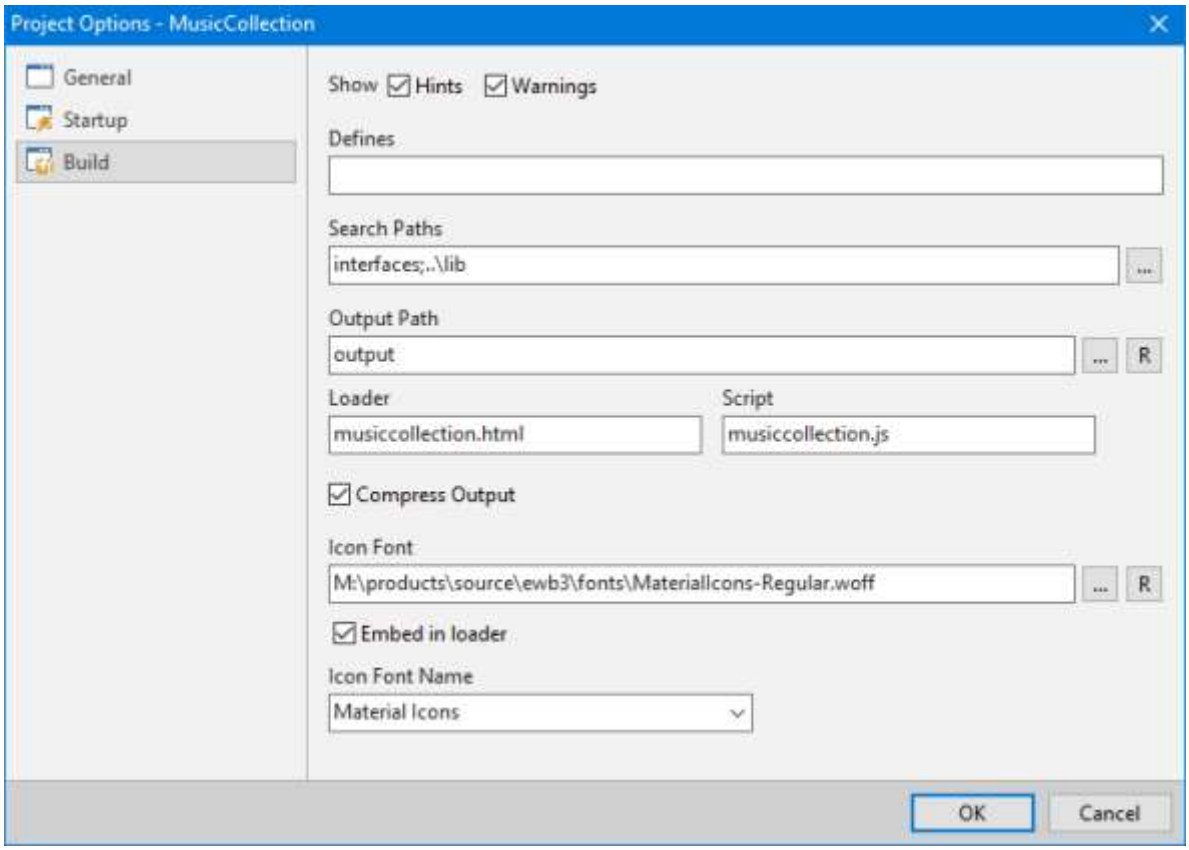
Option	Description

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

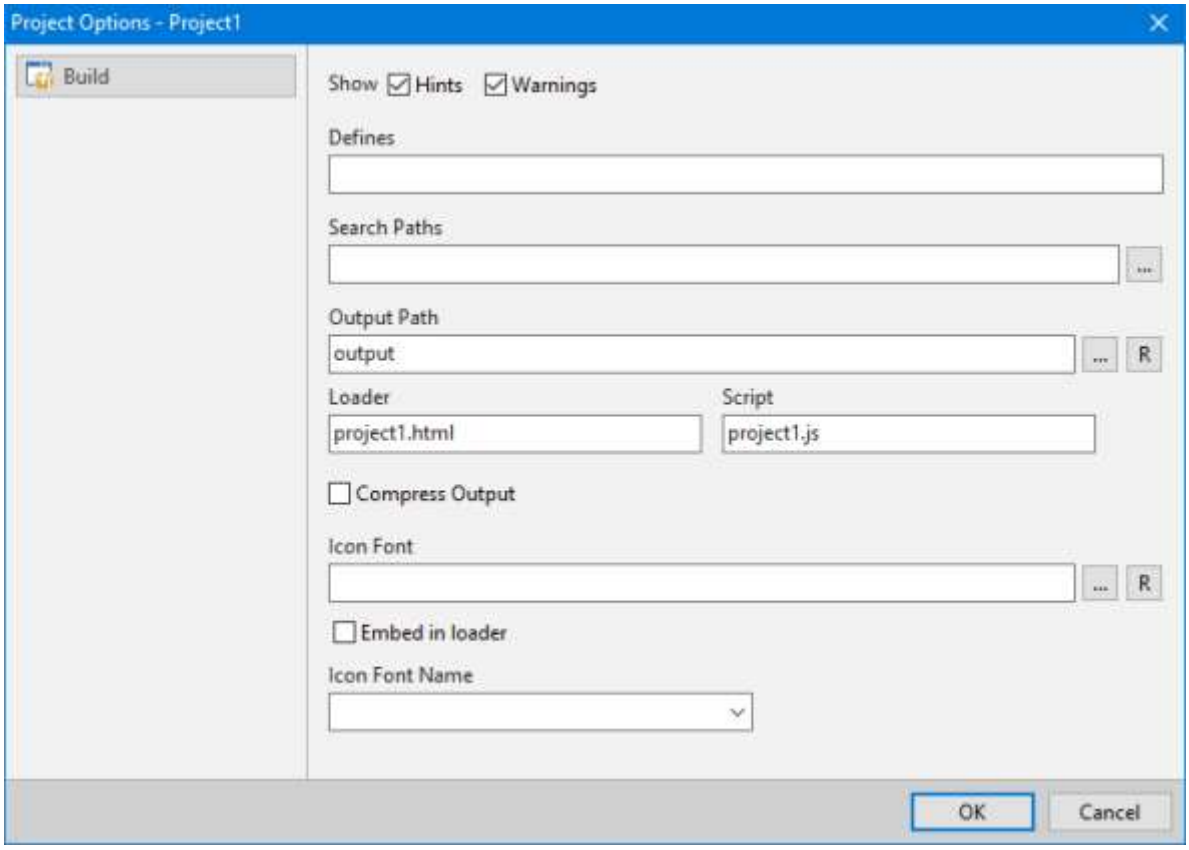
Build (Visual, Non-Visual, and Server)

For visual and non-visual client applications and server applications, the Build page allows you to configure the compilation options for the application.

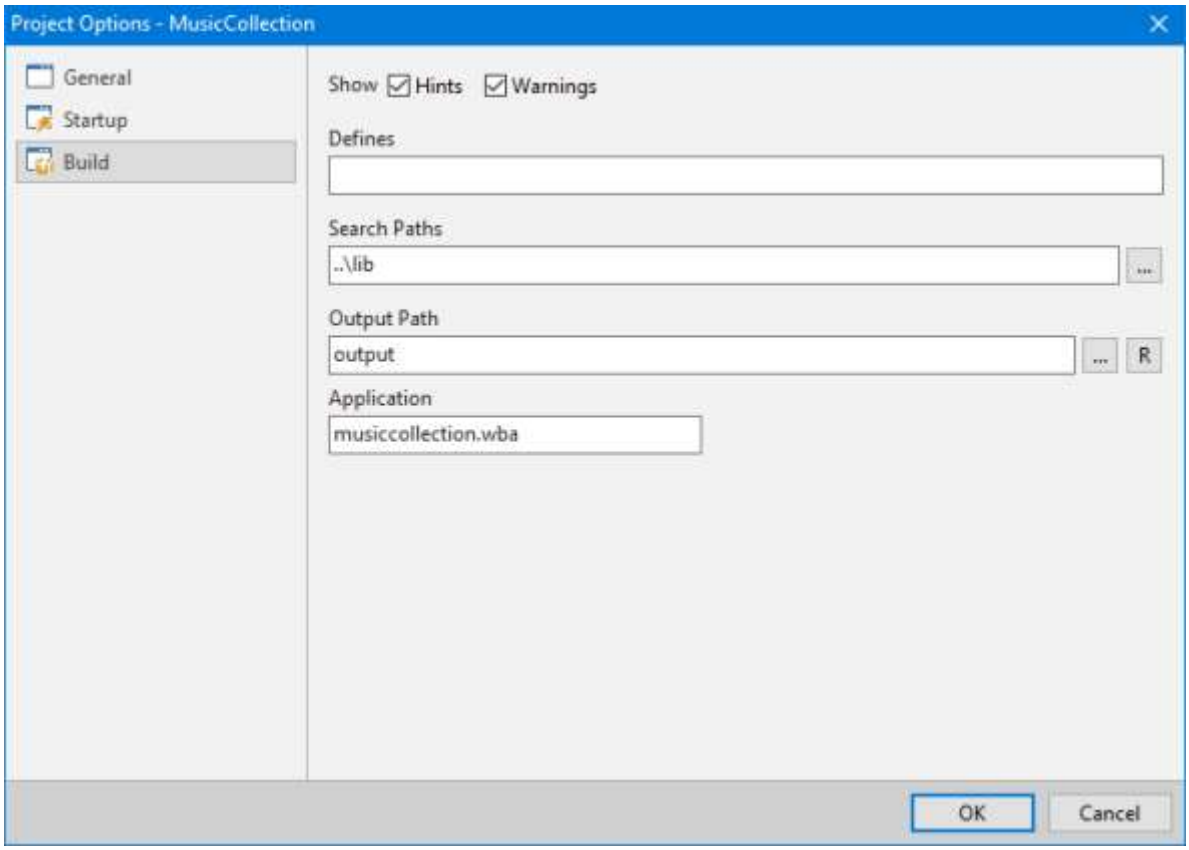
Visual client applications:



Non-visual client applications:



Server applications:



Option	Description
Show Hints/Warnings	Make sure these check boxes are selected (default) in order to see all hints and warnings from the compiler about unused variables and other compilation conditions that you may need to know about.
Defines	Enter any compiler defines that you wish to use to control how the source units in the project are compiled.
Search Paths	In many cases you will not need to include any additional compilation search paths for a project. By default, the compiler will look in the project source folder and the component library search paths for any contained or referenced units. Please see the Modifying IDE Settings topic for more information on modifying the component library search paths. However, in certain cases you may want to include additional search paths for common library source units or custom control interfaces that are used between multiple projects, and this is where you would do so. When specifying more than one search path, be sure to separate multiple paths with a semicolon (;). Both absolute and relative paths are accepted as search paths. Relative paths should be specified relative to the root project path.
Output Path	This path specifies the output path where the application HTML (.html) loader file and application JavaScript (.js) source files will be emitted for client applications, and where the application (.wba) file will be emitted for server applications. This path can be an absolute path or a relative path, and you can click on the relative path (R) button to convert an absolute output path to an output path that is relative to the root project path.
Loader	This file name specifies the emitted output name of the application loader file (.html) for client applications.
Script	This file name specifies the emitted output name of the application file (.js) for client applications.
Application	This file name specifies the emitted output name of the application file (.wba) for server applications.
Compressed Output	When this check box is selected, the compiler will emit the HTML and JavaScript files for a client application in a highly-compressed and obfuscated form. This normally can reduce the size of the emitted HTML and JavaScript files by 50% or more. **Note** Server applications use a custom binary format and are always compressed, so this options is unavailable for server application projects.
Icon Font	This file name specifies the icon font file to use for the icons in the icon library included in a client application. The icon font file name can use absolute or relative paths, but it is recommended that you use an absolute path in the file name so that there aren't any issues with the compiler finding the icon font file. You can click on the relative path (R) button to convert an absolute icon font file path to an icon font file path that is relative to the root project path. By default, the icon font file is set to the default icon font file EWBIcons located in the \fonts subdirectory under the main installation directory. Please see the Icon Library topic for more information on

	using icon fonts.
Embed in loader	This check box controls whether the specified icon font file name is embedded directly in the HTML loader file created when building a client application, or whether a link to the icon font file name is used instead. By default, the icon font file will be embedded in the HTML loader file.
Icon Font Name	This name specifies the font family name to use for the icon font file name when building a client application.

Please see the Building Applications topic for more information on building projects.

3.8 Using the Project Manager

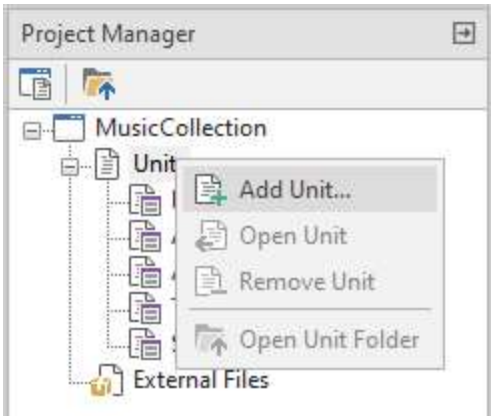
The project manager provides a quick and easy-to-use interface to the contents of a project, including all source units and external files such as images, fonts, and/or JavaScript source files.

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

Adding an Existing Source Unit to a Project

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

- Right-click on the **Units** node of the project contents tree. The context menu for the project manager will appear. Click on the **Add Unit** menu option.

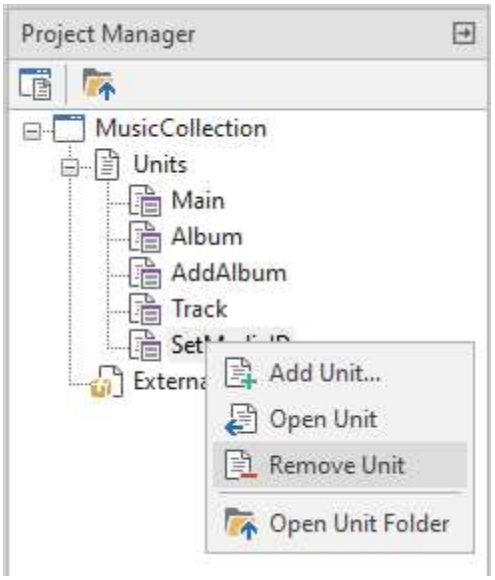


- A Windows file open dialog will appear. Navigate to, and select, the existing source unit that you wish to add to the project. Click on the **Open** button in the Windows file open dialog to complete adding the source unit to the project.

Removing a Source Unit from a Project

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

- Right-click on the name of the source unit that you wish to remove. The context menu for the project manager will appear. Click on the **Remove Unit** menu option.



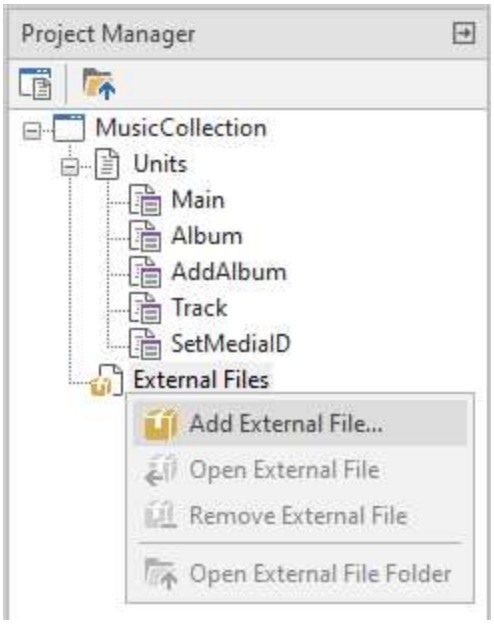
- A confirmation dialog will be displayed, asking you to confirm the removal of the source unit from the project. Click on the **Yes** button to continue, or the **No** button to cancel the removal.

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

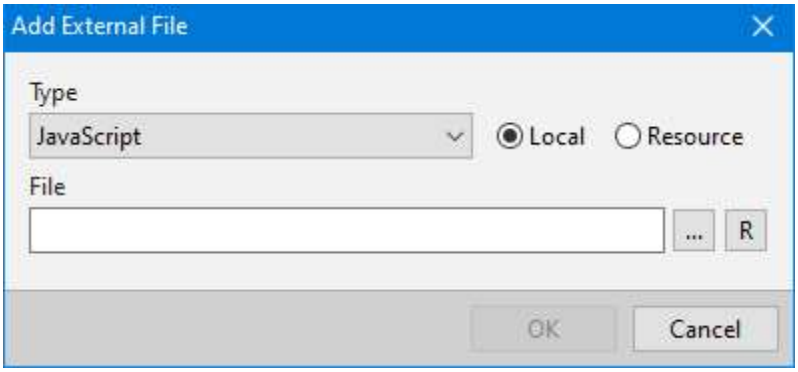
Adding an Existing External File to a Project

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

- Right-click on the **External Files** node of the project contents tree. The context menu for the project manager will appear. Click on the **Add External File** menu option.



- The Add External File dialog will appear:

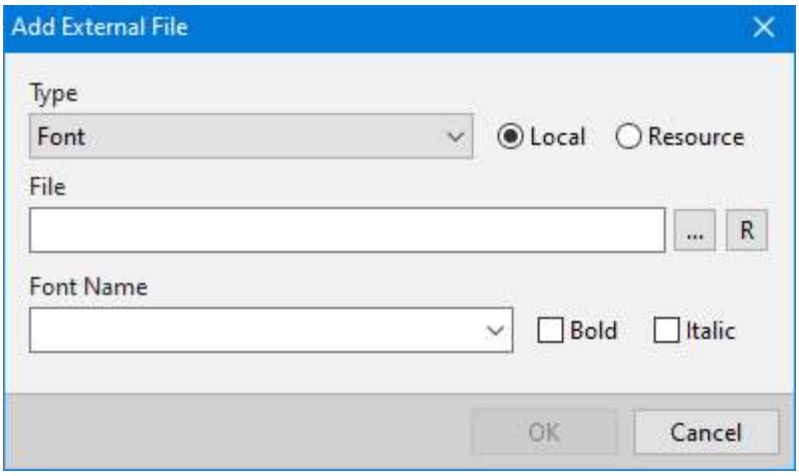


Select the type of external file to add using the Type combo box. If the file is a local file (the default), then leave the Local radio button selected and specify the local file name using the File edit control and/or the file selection button to the right of the edit control. If the file is an HTTP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

> **Note**
> When adding an external local file, the file name selected will contain an absolute path to the file. Click on the relative path (R) button to convert the file path to a path that is relative to the root project path.

External Font Files

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

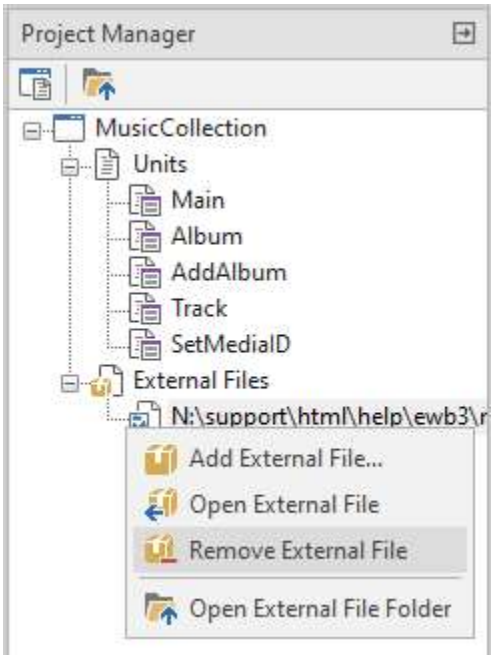


This additional information is necessary to ensure that the proper font linking information is added to the application's HTML loader file during compilation, and to ensure that the proper font is selected at runtime. Use the Font Name combo box to select or enter the name of the font that should be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

Removing an External File from a Project

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

- Right-click on the name of the external file that you wish to remove. The context menu for the project manager will appear. Click on the **Remove External File** menu option.



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

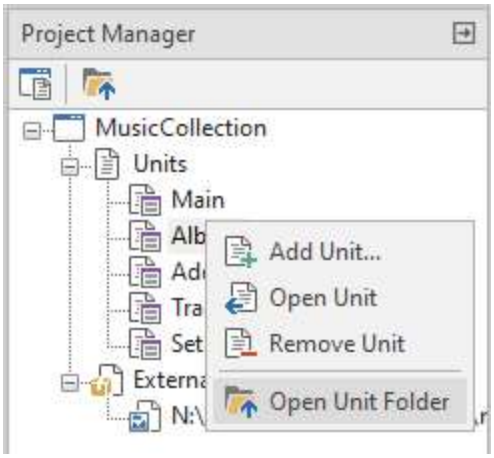
Note
Removing an external file from a project does **not** delete the actual external file on disk. It only removes the reference to the external file from the project configuration file (.wbc) so that it will no longer be deployed as part of the project.

Opening Folders for the Project Contents

You can open the folder for the project, any source units, or any external files so that you can browse the contents of the folder using the Windows Explorer.

Use the following steps to open a folder using the project manager:

- Right-click on the name of the project, a source unit, or an external file whose folder you wish to open. The context menu for the project manager will appear. Click on the **Open Project Folder** menu option for the project, the **Open Unit Folder** menu option for a source unit, or the **Open External File Folder** menu option for an external file.



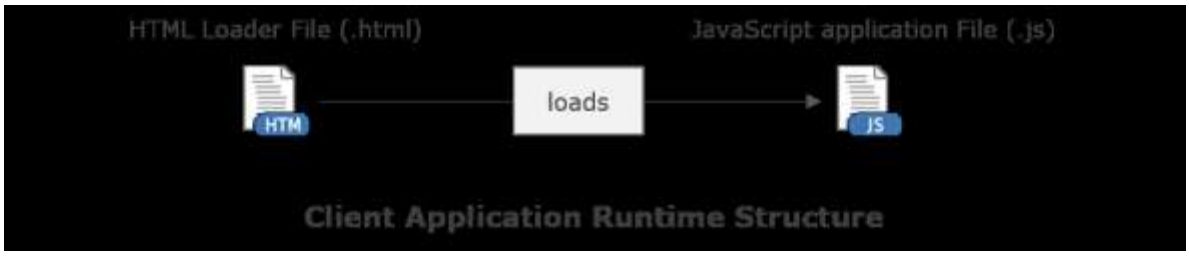
The folder for the project, source unit, or external file will be opened in a Windows Explorer window.

3.9 Building a Project

When a project is built, the compiler performs the following steps:

- The project source file is compiled.
- All source units referenced in the project source file are compiled. This includes both source units in the **contains** and **uses** clauses. In each source unit, all referenced source units are compiled, and this continues until all referenced source units are compiled.
- After all referenced source units are compiled, the application is emitted. For client applications, the compiler emits a single loader HTML file (.html) with the name specified by the output loader file name project option, and a single JavaScript file (.js) with the name specified by the output script file name project option. For server applications, the compiler emits a single server application file (.wba) with the name specified by the output application file name project option. All files are emitted to the output path specified by the output path project option. Please see the Modifying Project Options topic for more information on modifying the output path and output file names.
- Finally, any external files that have been included with the project are copied to the output path specified by the output path project option. Only external files that have changed since the last time the project was built are copied.

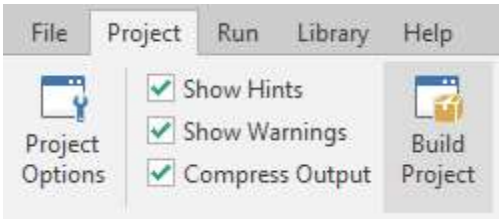
The following illustrates the emitted output of an application:



By default, all output files are emitted to path specified by the the output path project option. Please see the Modifying Project Options topic for more information on modifying the output path.

Use the following steps to build a project:

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



You can also use the keyboard to build an application by holding down the Ctrl key and pressing the **F9** key. You can use the Show Hints, Show Warnings, and Compressed Output check boxes on the Project menu to provide quick access to these build options for the project. They correspond to the same settings on the Build page of the Project Options dialog.

3.10 Deploying a Project

When a project is deployed, the IDE performs the following steps:

- The IDE authenticates against the target deployment server specified for the project. The default deployment server is the Internal web server built into the IDE. Please see the Using the Server Manager topic for more information on specifying the user ID to use for server management and deployment.
- The IDE starts to deploy the contents of the project's output path to the target deployment path on the deployment server. The target deployment path is a combination of a special base resource name followed by the deployment path specified for the project:

  ```
  <Resource Name>/<Project Deployment Path>
  ```

  For client applications, the output files are uploaded to the **Content** resource on the deployment server. For server applications, these files are deployed to the **Applications** resource on the deployment server. Please see the Configuring the Web Server topic for more information on how these resources are defined on the deployment server.

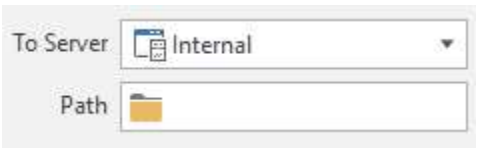
  > **Note**
  > If any part of the deployment path doesn't exist, it will automatically be created. If the user account ID used for the deployment does not have the proper privileges for creating folders on the deployment server, then the deployment will fail with a privileges error. Please see the Using the Server Manager topic for more information on specifying the user ID to use for server management and deployment.

- As each project output file is uploaded to the deployment server, the IDE first checks to see if the file already exists in the deployment path. If the file already exists and hasn't been modified, then the file upload is skipped for that file. This saves a lot of time when some of the project output files are large external files being included with the project. Please see the Using the Project Manager topic for more information on including external files with a project.

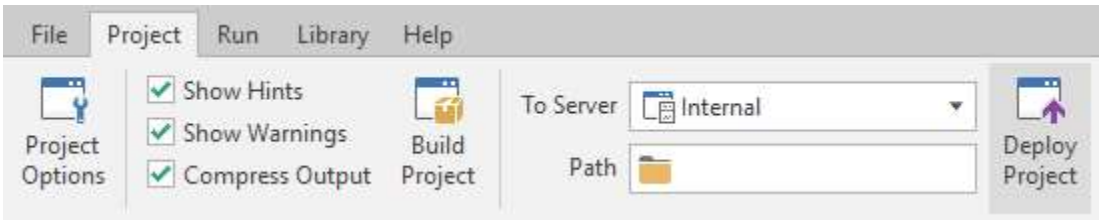
  > **Note**
  > The last compiled version of a project application will be uploaded to the deployment when a project is deployed. It is always wise to make sure to build the project before deploying in order to ensure that the most recent version of the application is uploaded.

Use the following steps to deploy a project:

- Click on the **Project** tab on the main menu.
- Select the deployment server from the list of available servers and modify the deployment path as necessary.



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



You can also use the keyboard to deploy an application by holding down the **Alt** key and pressing the **F9** key.

3.11 Running and Debugging a Project

It is possible to run both client applications and server applications directly in the IDE for testing purposes. For client applications, the IDE uses an embedded version of the Internet Explorer web browser to run the application.

> **Note**
> The IDE requires that Internet Explorer 11 or higher be installed in order to properly run applications in the IDE.

For server applications, the IDE uses a request selected from the requests defined in the Request Manager in the IDE. Please see the Using the Request Manager topic for more information on defining requests to use when running server applications in the IDE.

> **Note**
> Server applications can be debugged while they are being run in the IDE, but not client applications. Once Internet Explorer 11 is replaced with an embedded version of Microsoft Edge in a future release, debugging functionality for client applications will also be made available in the IDE.

Use the following steps to run a project:

- Click on the **Run** tab on the main menu.
- For client applications, select the run server from the list of available servers, select whether or not to automatically deploy the application before running it, and modify the run parameters as necessary.

> **Note**
> The Internal web server built into the IDE always uses automatic deployment of an application.

Parameters are specified in the following format:

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

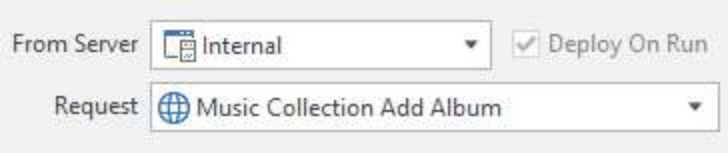
You can also specify an anchor to be used with the URL used to run the application:

```
#anchor1
```

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

The complete target URL that the IDE uses to run a client application is determined by the base URL for the run server, the deployment path for the project, and the specified parameters:

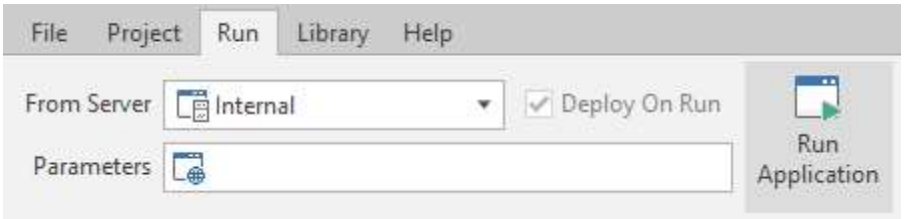
- For server applications, select the run server from the list of available servers, select whether or not to automatically deploy the application before running it, and select the request to use for running the application.



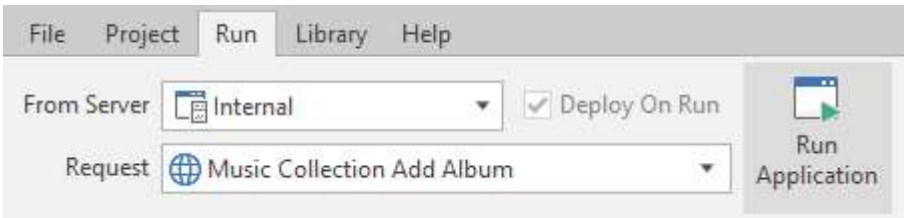
> **Note**
> The Internal web server built into the IDE always uses automatic deployment of an application. Also, the complete definition of the HTTP request used to run the server application is contained within the selected request. Please see the Using the Request Manager topic for more information on defining requests to use when running server applications in the IDE.

- Click on the **Run Application** button on the Run menu to run the current project. If necessary, the IDE will automatically build the application before running it. If any errors are present, the build will fail and the application will not run. If the Deploy On Run option has been selected, the application will be automatically deployed after it has been successfully built, but before it is run. If there are any errors during deployment, or if the deployment is cancelled, then the application will not run.

  Client applications:



  Server applications:



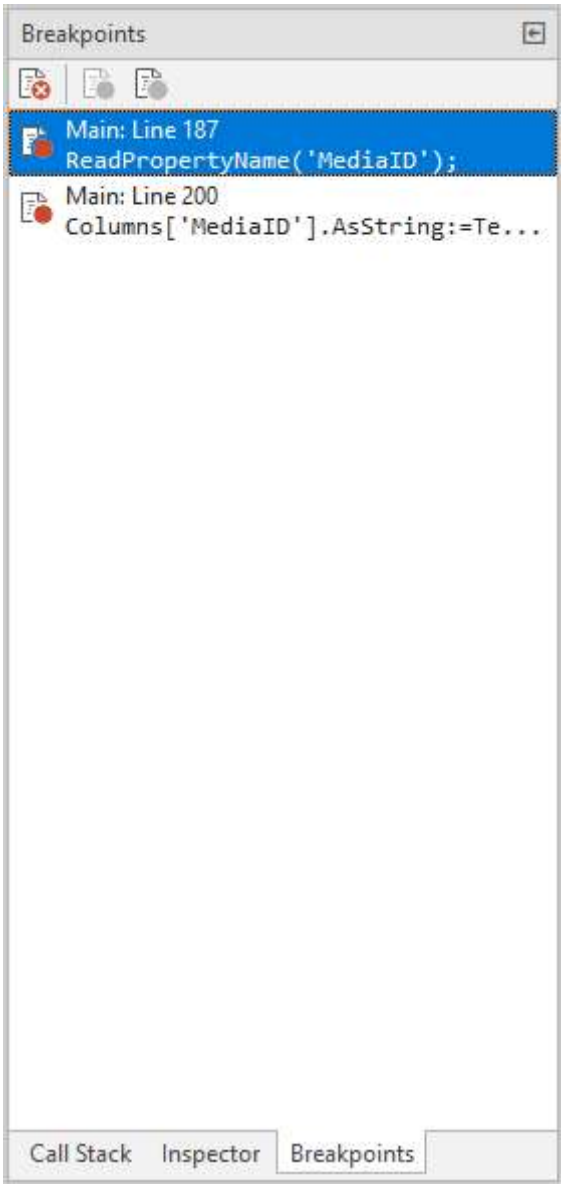
  You can also use the keyboard to run an application by pressing the **F9** key.

Debugging Server Applications

Both before and while running a server application in the IDE, you can define source unit breakpoints so that the application execute will pause when it hits such a breakpoint. Breakpoints can be added by clicking to the left of the line numbers in the gutter area of the code editor:

You can also toggle a breakpoint for the current line in the code editor by pressing the **F5** key.

Once a breakpoint has been set, it is added to the breakpoints:

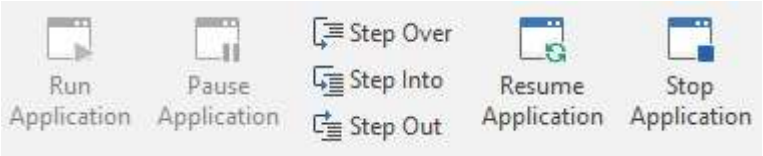


You can navigate to a breakpoint by double-clicking on the breakpoint or by pressing the **Enter** key when the breakpoint is selected. In order to delete, enable, or disable a given breakpoint, you can use the breakpoints toolbar or you can right-click on the breakpoint and select the desired option from the context menu. You can also delete a breakpoint by selecting the breakpoint and pressing the **Del** key.

> **Note**
> Disabling a breakpoint simply prevents it from pausing the execution of the application, whereas deleting a breakpoint removes the breakpoint completely.

Once a server application has been successfully started on the run server, the interactive debugging functionality will become active in the IDE. If any breakpoints were set in the source units for the project, these breakpoints will be updated to reflect whether they are valid for the compiled application present on the run server. If any valid breakpoints are encountered while the application is being executed on the run server, the application execution will pause and various debugger command buttons will be enabled.

Debugger Commands



Click on the **Step Over** button to step over the current source unit line being executed, even if the line contains a function or procedure call. You can also use the keyboard to step over a source unit line by pressing the **F8** key.

Click on the **Step Into** button to step over the current source unit line being executed if the line does not contain a function or procedure call, or step into the corresponding function or procedure if the line does contain such a call. The debugger will step into the first call followed by any other calls on the same line, from left to right. You can also use the keyboard to step over a source unit line by pressing the **F7** key.

Click on the **Step Out** button to step out of the current procedure or function body. You can also use the keyboard to step out of a procedure or function body by holding down the **Shift** key and pressing the **F7** key.

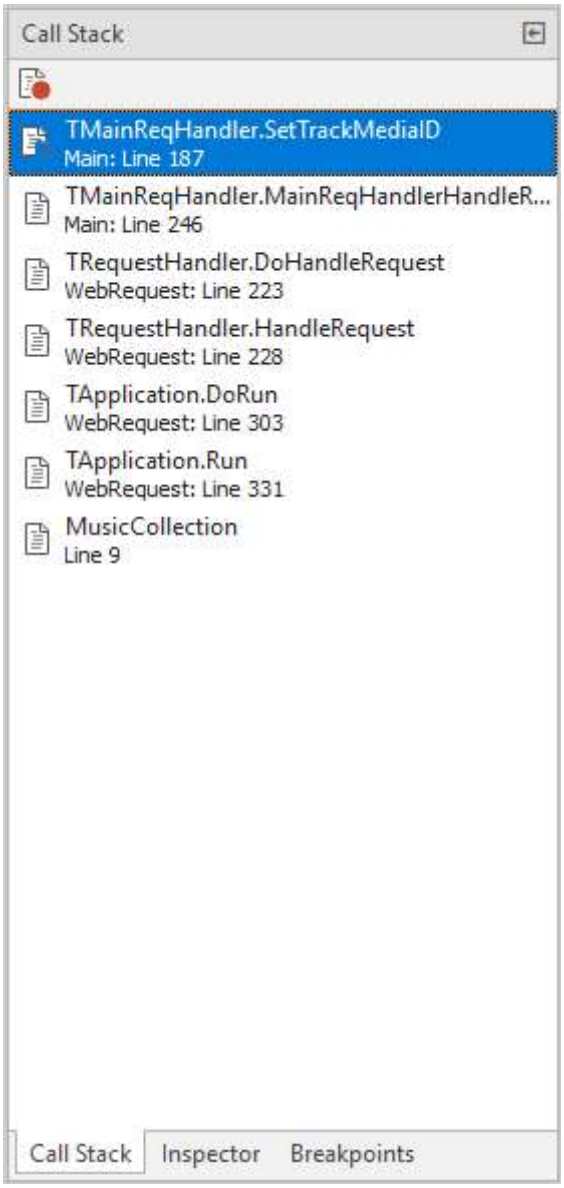
Click on the **Resume Application** button to resume execution of the application. The application execution will then continue until it either hits another breakpoint, is stopped, or completes. You can also use the keyboard to resume execution by pressing the **F9** key.

Click on the **Stop Application** button to stop the execution of the application. You can also use the keyboard to stop the execution by holding down the **Ctrl** key and pressing the **F2** key.

Whenever the execution of the application is paused, you will also see two new tabs appear in the Tools area of the IDE: the call stack and the local variable inspector.

Call Stack

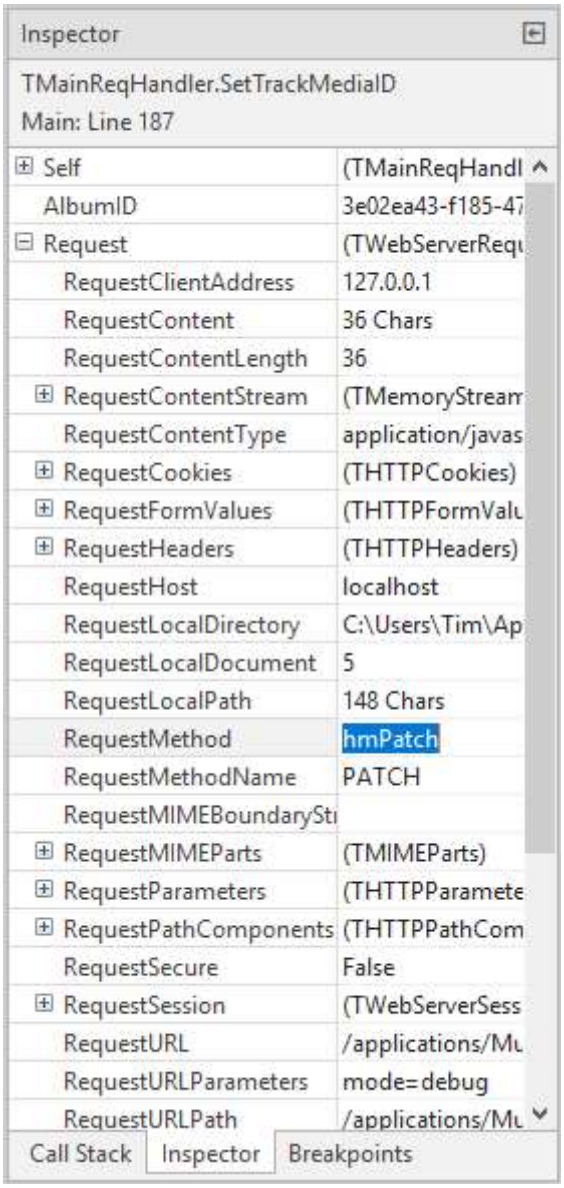
The call stack will show all of the calls that preceded the current breakpoint, in the reverse order of when the call was executed:



The call stack allows you to navigate to a call by double-clicking on the call or by pressing the **Enter** key when the call is selected. In order to toggle a breakpoint for a given call, you can use the call stack toolbar or you can right-click on the call and select the Toggle Breakpoint option from the context menu.

Local Variable Inspector

The local variable inspector allows you to examine the value of any local variables, including the properties of class instances.



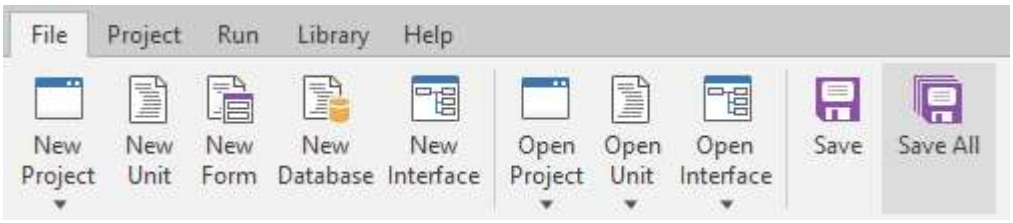
Note
For any variables or properties that use special dialogs for displaying the contained value, you will see a "..." button next to the property or variable. Clicking on this button will launch the dialog.

3.12 Saving a Project

Saving Projects

Use the following steps to save a project:

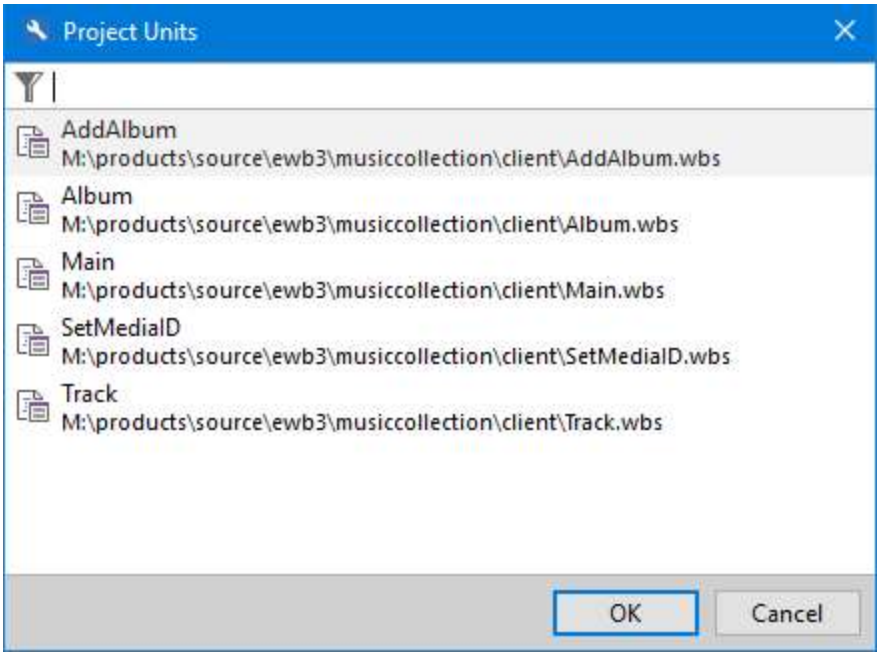
- Click on the **File** tab on the main menu.
- Click on the **Save All** button on the File menu to save any changes to the project and other open source units.



3.13 Searching for Project Units

Use the following steps to access a list of all contained source units in a project.

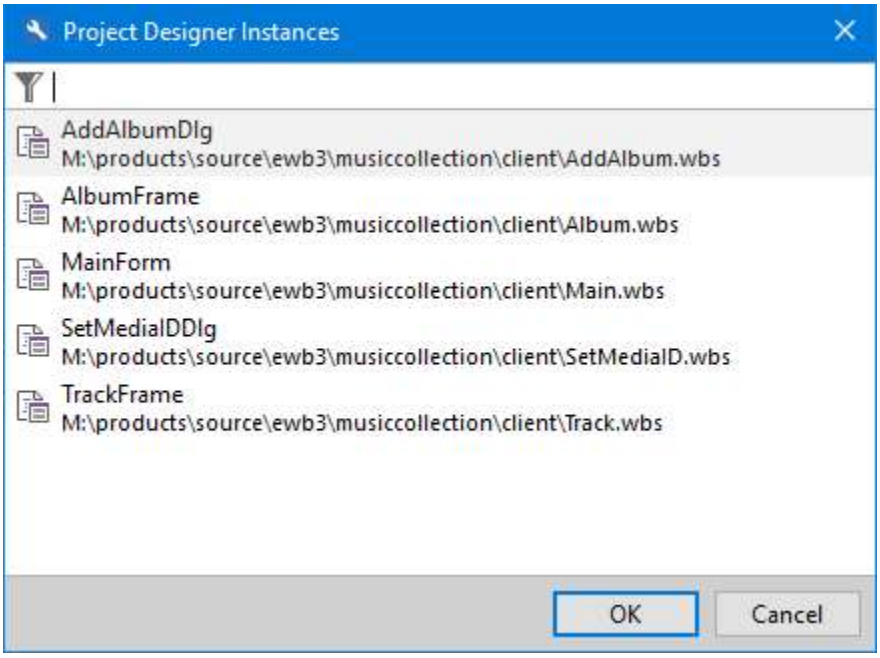
- Hold down the **Ctrl** key and press the **F12** key. The **Project Units** dialog will appear, allowing you to filter the project units by name and select a unit to open in the IDE.



- Click on the **OK** button to open the unit and any associated designer instance in the IDE.

Use the following steps to access a list of all designer instances associated with the project units. For visual client application projects, this includes forms and databases. For server application projects, this includes request handlers and databases.

- Hold down the **Shift** key and press the **F12** key. The **Project Designer Instances** dialog will appear, allowing you to filter the designer instances by name and select a designer instance to open in the IDE.

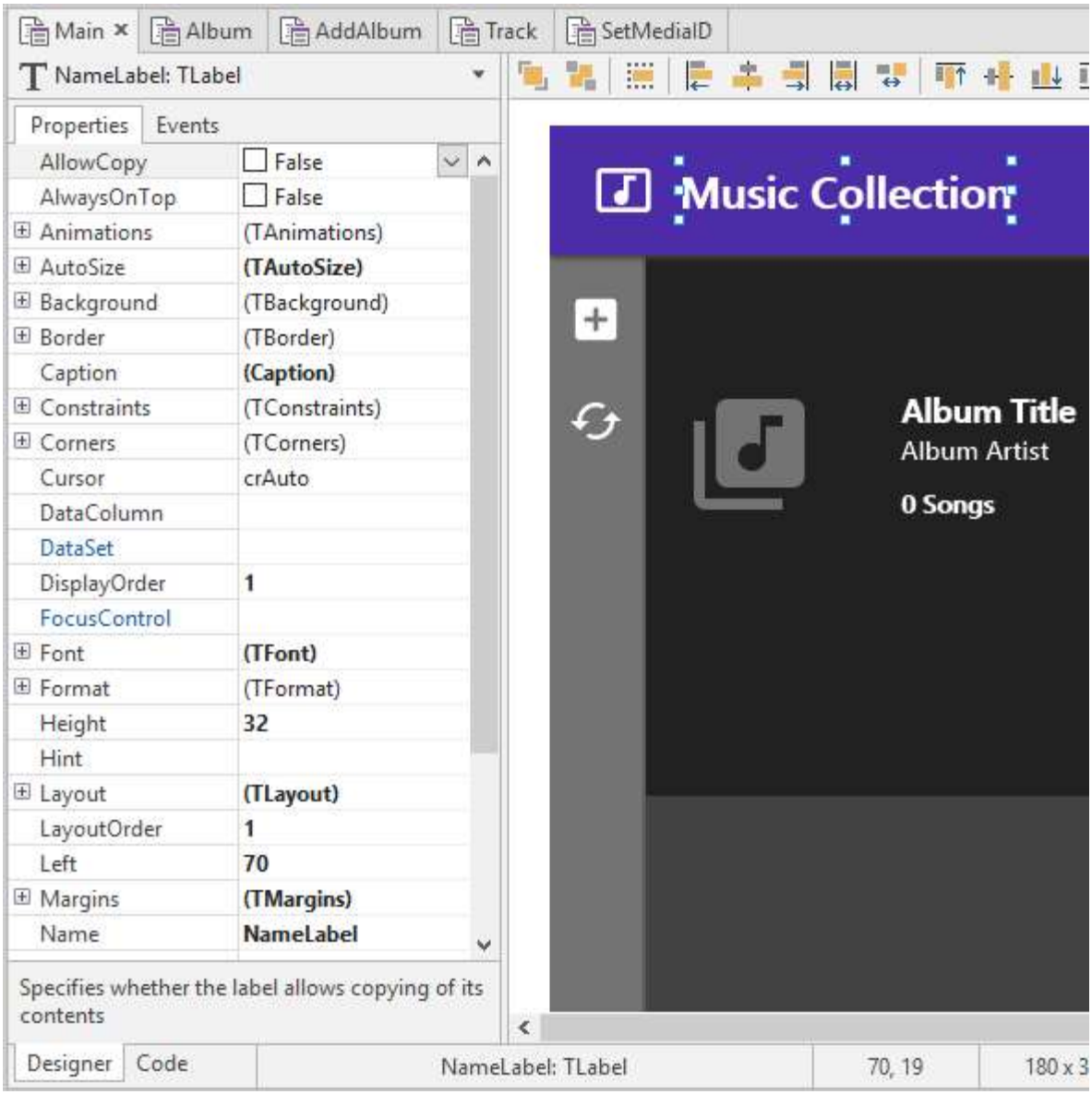


- Click on the **OK** button to open the designer instance and its associated unit in the IDE.

Note
Project units that don't have associated designer instances will not appear in the Project Designer Instances dialog.

3.14 Using the Component Inspector

The component inspector allows you to modify various properties of controls and non-visual components that present on the designer surface in the work area.

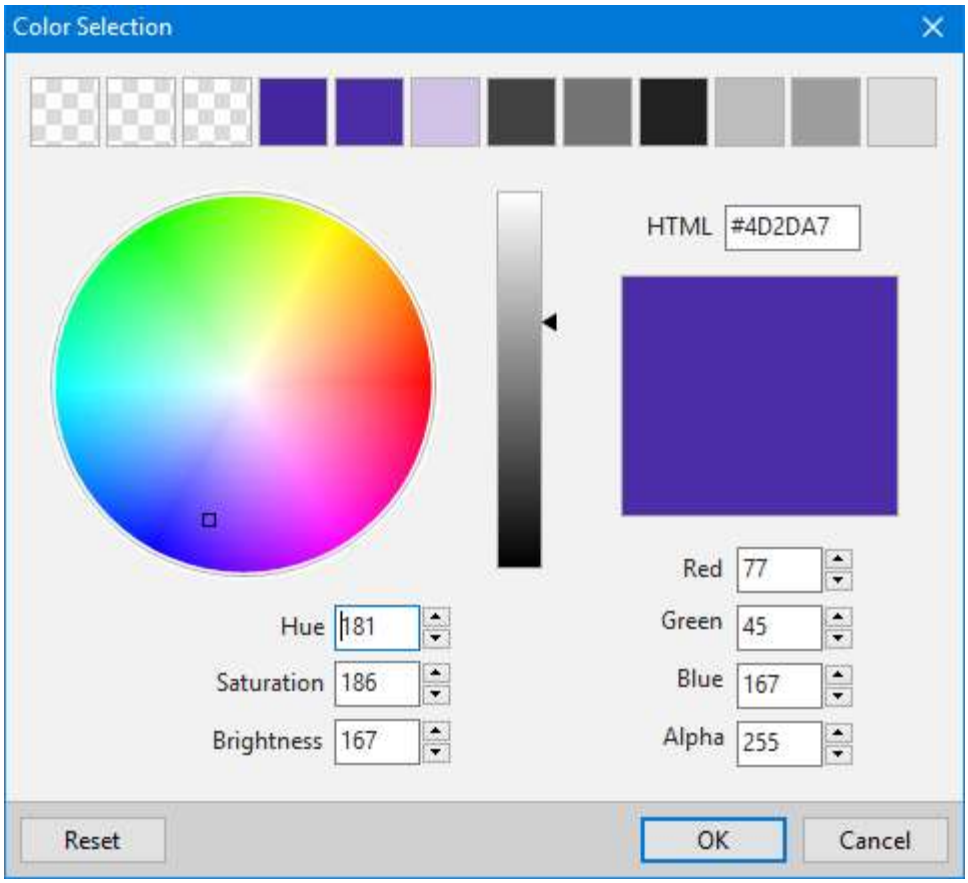


Note
The component inspector will only show properties for the currently-selected components on the active designer surface in the work area. It will be blank if a designer surface is not active, such as when you are editing a source unit that does not have an associated form, database, or request handler,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

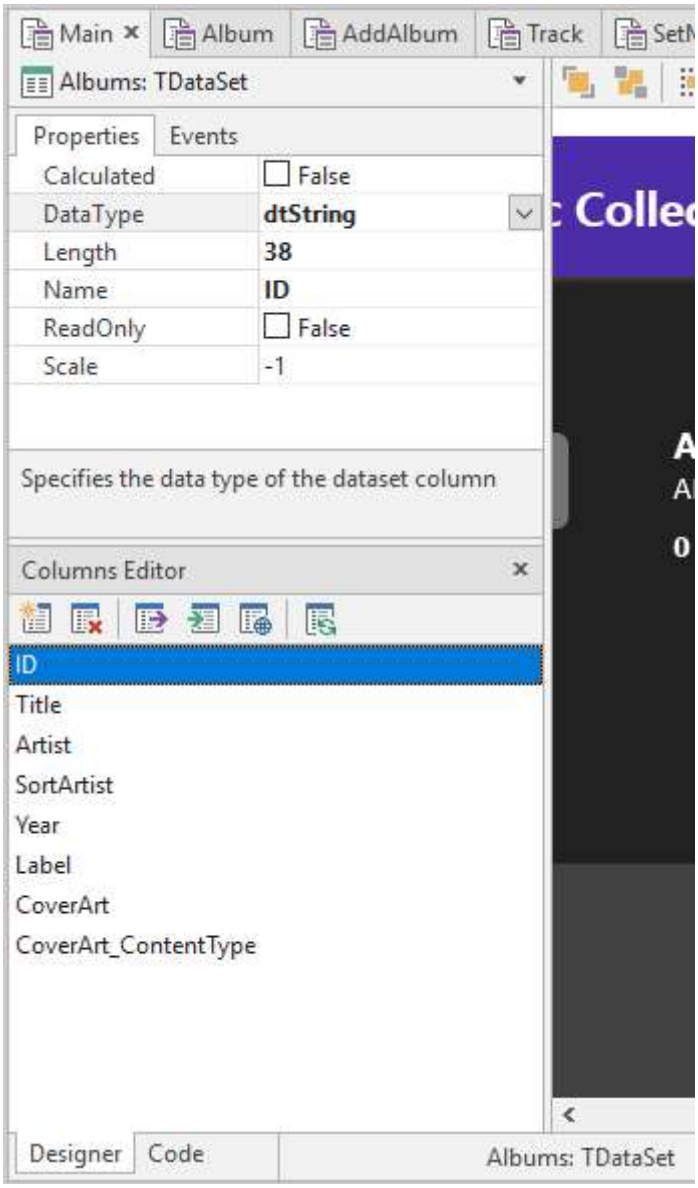
The component inspector consists of a component selection combo box and two pages that represent the properties and events of a component. You can switch between the two by clicking on the appropriate tab at the top of the component inspector.

Modifying Properties

To modify any property of a component, make sure that the **Properties** page is the active page in the component inspector, click on the desired property value, and type in the new value. If applicable, the property may have a special property editor in the form of a drop-down list or dialog that is accessible by clicking on the (...) button to the right of the property value or by double-clicking on the property value. The following is an example of the Color property editor for the TLabel Background Fill property:



Properties that represent collections, such as the TDataSet Columns property, will cause an applicable collection editor to be launched below the component inspector:



Basic information about each property can be found at the bottom of the component inspector. Any properties whose value has been changed from the default value will be indicated by the value being displayed in a bold font.

Modifying Event Handlers

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

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

To delete an existing event handler, but keep the event handler code present in the source unit, clear out the event

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

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

Context-Sensitive Help

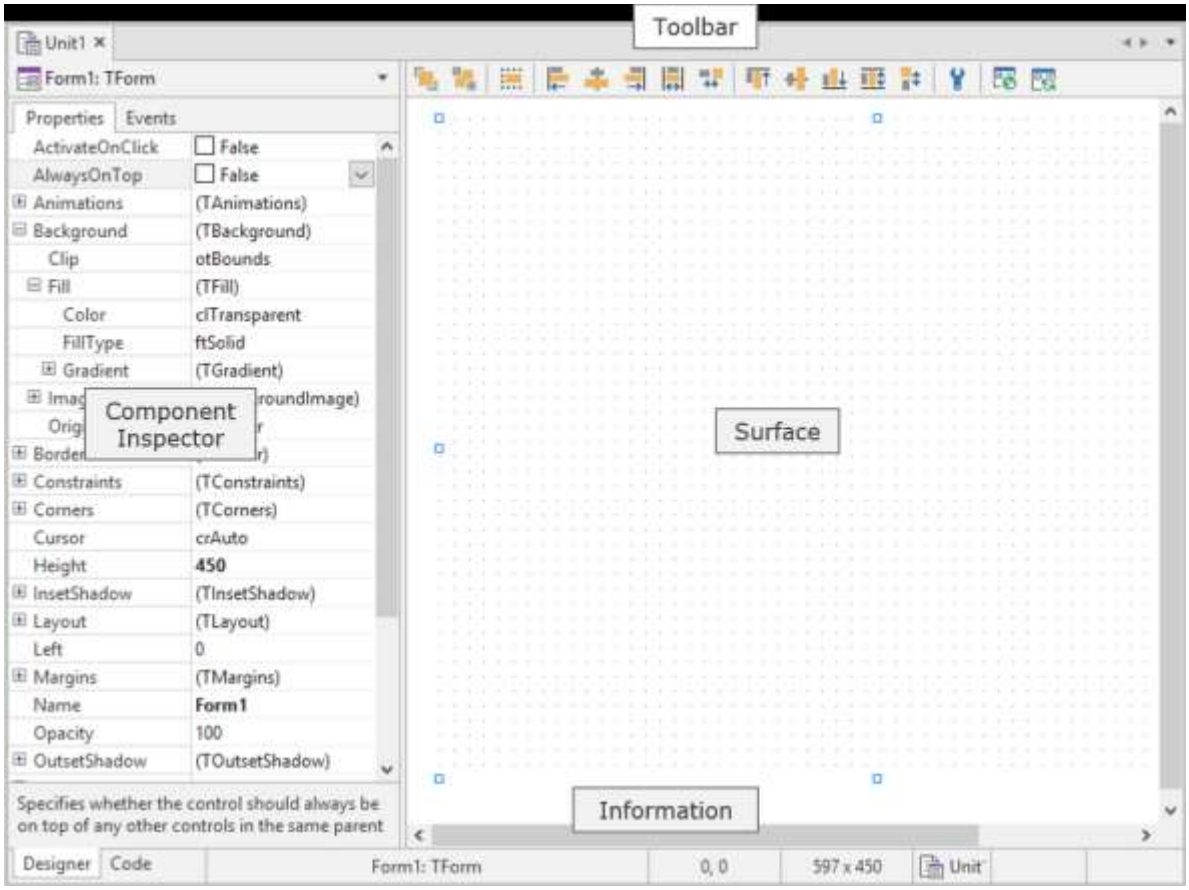
You can get context-sensitive help on any property or event in the component inspector by clicking on the desired property or event and pressing the **F1** key. For more information on using the help browser, please see the Accessing Help topic.

3.15 Using the Designer

The designer surface in the work area allows the developer to visually manipulate the controls and components contained within a designer instance. For visual client applications, designer instances can be a form or database. For server applications, designer instances can be a request handler or database.

> **Note**
> Request handlers and databases are inherently non-visual and their visual size is exclusively a design-time property that is not used at runtime.

The designer surface has the following layout:



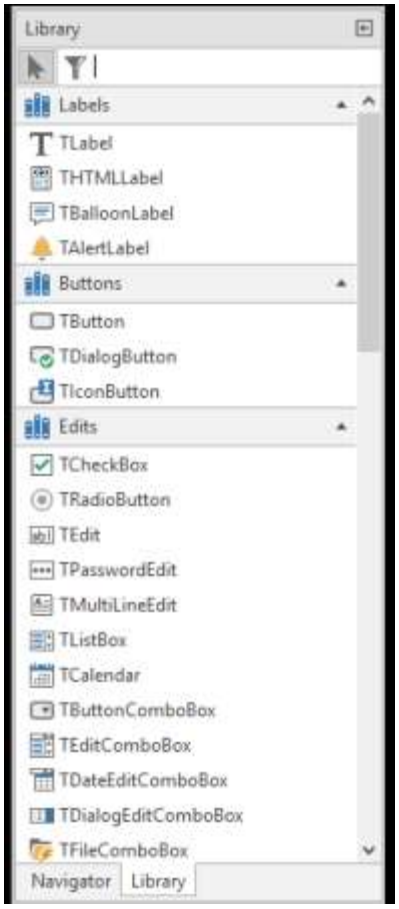
The unit of measure used by the designer surface is the pixel, and the resolution is always assumed to be 96 pixels per inch. All modern web browsers use a virtual resolution of 96 pixels per inch, regardless of the actual resolution on the client machine's display. The web browser automatically handles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

By default, the designer surface shows a grid to aid with component placement and alignment, and the grid guides (dots) are spaced apart at 10 pixel intervals. Please see the Modifying IDE Settings topic for more information on modifying the designer grid properties.

Note
Any time you hover the mouse over any component on the designer surface, the name, position, and size of the component will be displayed in the information panels at the bottom of the designer.

Adding a Control or Component to a Form

When a designer is active in the IDE, the component library is accessible via the Tools window in the IDE:



The component library is used to add both visual controls and non-visual components to the active designer. Non-visual components are represented visually at design-time, but are not visible at runtime.

Note
Request handlers and databases are inherently non-visual and only allow non-visual components such as the TServerRequest and TDataSet components to be dropped on to the designer surface.

To filter the components in the component library, simply type in the filter text in the edit control above the component library. This is useful when there are many installed components and you're not sure of the category where the component was installed.

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

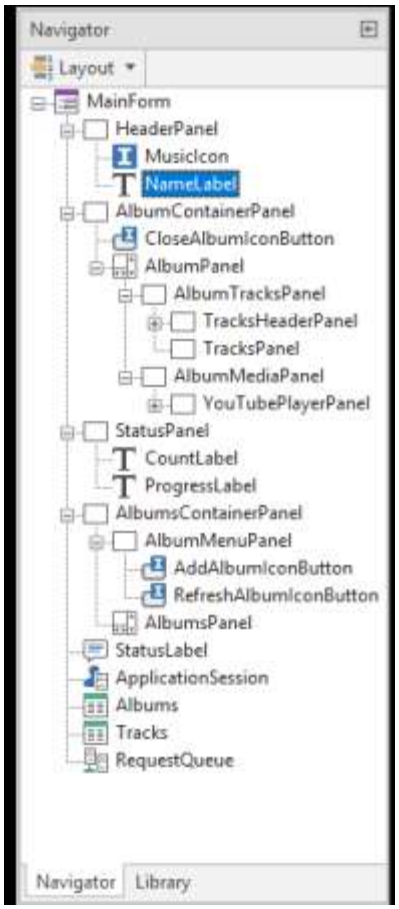
To add a control or non-visual component to the active designer, click on the desired control or component in the

component library and drag it on to the active designer surface. For controls, you can also click on the control in the component library and then draw the control on to the active designer surface using the mouse. This is useful when you want to resize the control to a particular size when it is first added to the designer surface.

> **Note**
> When adding non-visual components to a form, the non-visual components are always added directly to the form and are never contained within another control. If you drop a non-visual component on a container control, it will be added directly to the form.

Selecting a Control or Component

When a designer is active in the IDE, the component navigator is accessible via the Tools window in the IDE:



To select a single control or component via the component navigator or the designer surface, click on the desired control or component with the left mouse button. To select more than one control or component, hold down the **Ctrl** key while clicking on the desired controls and components with the left mouse button. Selecting multiple controls and components is desirable when one wants to resize or align multiple controls at the same time to ensure that their placement or size is uniform, or when one wants to copy and paste a group of controls or components. Multiple control and component selection is also useful for setting a property to the same value for multiple controls or components using the component inspector.

> **Note**
> You can only select multiple controls or components within the same parent control. For non-visual components, this parent control would be the form, database, or request handler designer instance.

You can also use the mouse to directly select a group of controls or components on the designer surface using a lasso:

- If the controls or components are placed directly on a form, database, or request handler, you can click and hold down the left mouse button to begin the selection. Then, while keeping the left mouse button down, move the mouse to lasso the desired control(s) or component(s).
- If the controls are in a container control such as a panel, you can click and hold down the left mouse button, while also pressing the **Ctrl** key, to begin the selection. Then, while keeping the left mouse button and **Ctrl** key down, move the mouse to lasso the desired control(s).

Resizing a Control

Once a control or component has been placed on the designer surface, you will see that the control or component will have designer handles on all four sides and corners:

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

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

Moving a Control or Component

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

Control Layout and Alignment

The layout toolbar for the designer can be used to adjust the display order (send to back/bring to front) and position of controls on the active designer:

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

Deleting a Control or Component

To delete a control or component, select the desired control or component using the component navigator or designer surface and press the **Del** key. This will also work when multiple controls/components are selected.

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

Default Event Handlers

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

Toggling Between the Code Editor and Designer

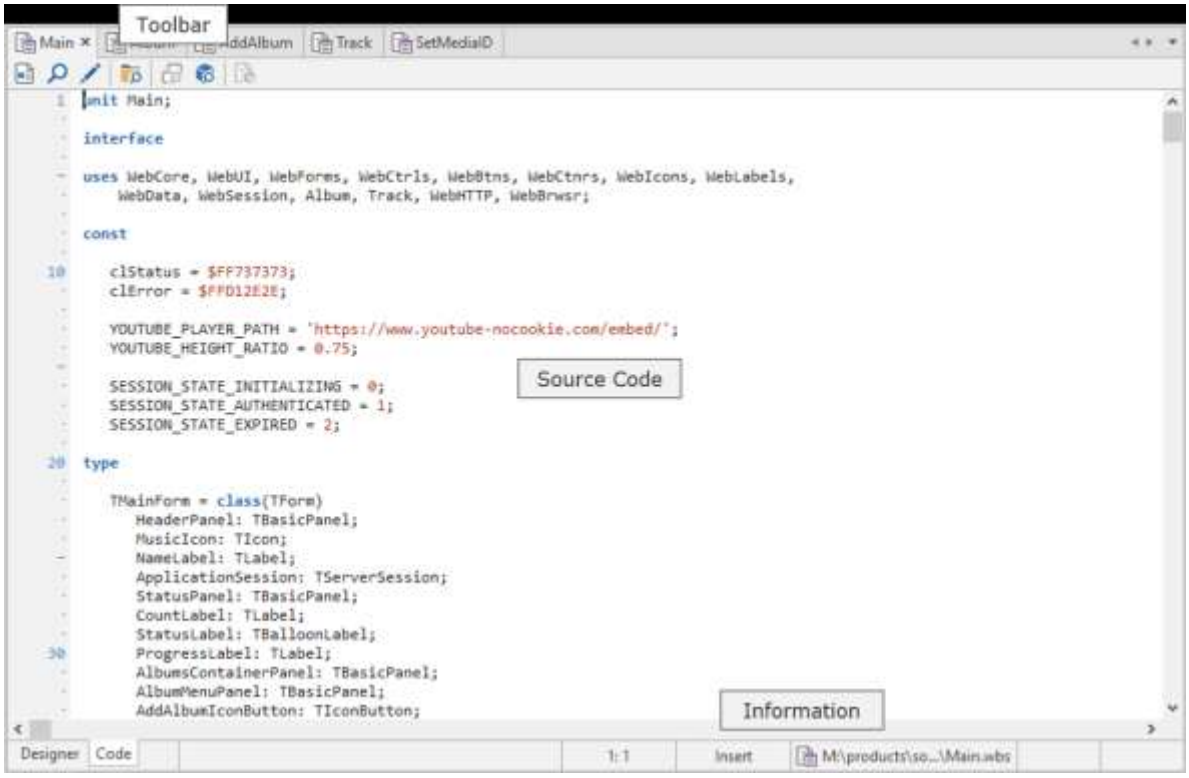
In order to toggle between the code editor and the designer, press the **F12** key or click on the Code or Designer tabs in the lower left-hand corner of the work area in the IDE.

Context-Sensitive Help

You can get context-sensitive help on any control or component in the designer by selecting the desired control or component and pressing the **F1** key. For more information on using the help browser, please see the Accessing Help topic.

3.16 Using the Code Editor

While the designer handles the visual layout of controls in a visual client application, the code editor is where the actual functionality behind a form, database, or request handler is implemented. The code editor has the following layout:



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

Automatic Code Updates

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

```
unit Unit1;

interface

uses WebCore, WebUI, WebForms, WebCtrls;

type

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

var
   Form1: TForm1;

implementation

end.
```

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

```
unit Unit1;

interface

uses WebCore, WebUI, WebForms, WebCtrls, WebBtns;

type

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

var
   Form1: TForm1;

implementation

end.
```

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

```
unit Unit1;

interface

uses WebCore, WebUI, WebForms, WebCtrls, WebBtns;

type

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

var
   Form1: TForm1;

implementation

procedure TForm1.Button1Click(Sender: TObject);
begin

end;

end.
```

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

```
unit Unit1;

interface

uses WebCore, WebUI, WebForms, WebCtrls, WebBtns;

type

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

var
   Form1: TForm1;

implementation

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

end.
```

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

Toggling Between the Code Editor and Designer

In order to toggle between the code editor and the designer, press the **F12** key or click on the Code or Designer tabs in the lower left-hand corner of the work area in the IDE.

Key Mappings

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

Keys	Action
Up Arrow	Moves the cursor to the previous line in the source code.
Down Arrow	Moves the cursor to the next line in the source code.
Page Up	Moves the cursor to the previous page in the source code.
Page Down	Moves the cursor to the next page in the source code.
Home	Moves the cursor to the start of the current source code line.
End	Moves the cursor to the end of the current source code line.
Left Arrow	Moves the cursor to the previous character on the current source code line.

Right Arrow	Moves the cursor to the next character on the current source code line, or to the next line if at the end of the current source code line.
Enter	Inserts a new line at the current cursor position.
Insert	Toggles the insert/overwrite mode for the keyboard.
Shift-Insert	Pastes the source code contents of the clipboard, if any, into the current cursor position.
Delete	Deletes the character at the cursor position.
Shift-Delete	Copies the currently-selected source code to the clipboard and deletes the source code from the source code ("cut" operation).
Backspace	Deletes the character right before the cursor position. If the cursor is at the start of a source code line, then the current source code line is moved to the end of the previous source code line (if present).
Tab	Inserts <tab size> spaces at the current cursor position, if the keyboard is in insert mode, or moves the current cursor position by <tab size> spaces if the keyboard is in overwrite mode. Please see the Modifying IDE Settings topic for more information on modifying the tab size used by the code editor.
Shift-Tab	Removes <tab size> spaces working back from the current cursor position, if the keyboard is in insert mode, or moves the current cursor position to the left by <tab size> spaces if the keyboard is in overwrite mode. Please see the Modifying IDE Settings topic for more information on modifying the tab size used by the code editor.
Ctrl-Home	Moves the cursor to the first source code line.
Ctrl-End	Moves the cursor to the last source code line.
Ctrl-Page Up	Moves the cursor to the first visible line on the current page.
Ctrl-Page Down	Moves the cursor to the last visible line on the current page.
Ctrl-Left Arrow	Moves the cursor to the start of the previous word in the source code.
Ctrl-Right Arrow	Moves the cursor to the start of the next word in the source code.
Ctrl-Up Arrow	Scrolls the code editor window up by one line.
Ctrl-Right Arrow	Scrolls the code editor window down by one line.
Ctrl-Enter	Opens the unit name or control interface name at the cursor position. If text is selected, then the selected text will be used first for searching for a valid unit name or control interface. If no text is selected, or the selected text does not represent a valid unit or control interface name, then the editor will use the identifier at the cursor position.
Ctrl-Space	Manually invokes code assistance at the cursor position. By default, code assistance is enabled so that it is automatically invoked after a specified delay (default 400 msecs). However, if code assistance is disabled, then you can still access the code assistance using manual key combination. Please see the Modifying IDE Settings topic for more information on enabling or disabling the code assistance in the code editor.
Ctrl-/	Comments and un-comments (toggle) the current source code line.

Ctrl-Insert	Copies the currently-selected source code to the clipboard.
Ctrl-Backspace	Deletes the word right on, or right before, the current cursor position. If the cursor is at the start of a source code line, then the current source code line is moved to the end of the previous source code line (if present).
Ctrl-A	Selects all source code in the code editor.
Ctrl-C	Copies the currently-selected source code to the clipboard.
Ctrl-I	Indents the current source code line by <tab size> spaces. Please see the Modifying IDE Settings topic for more information on modifying the tab size used by the code editor.
Ctrl-N	Inserts a new line at the current cursor position.
Ctrl-T	Deletes the word at the current cursor position.
Ctrl-U	Un-indents the current source code line by <tab size> spaces. Please see the Modifying IDE Settings topic for more information on modifying the tab size used by the code editor.
Ctrl-V	Pastes the source code contents of the clipboard, if any, into the current cursor position.
Ctrl-X	Copies the currently-selected source code to the clipboard and deletes the source code from the source code ("cut" operation).
Ctrl-Y	Deletes the current source code line.
Ctrl-Shift-Y	Deletes the source code from the current cursor position to the end of the current source code line.
Ctrl-Z	Reverses the last edit or find operation performed on the source code ("undo" operation).
Ctrl-Shift-Z	Replays the last edit or find operation on the source code that was reversed ("redo" operation).
Ctrl-Shift-Down Arrow	Moves the cursor from the class definition of a method to the implementation of the method.
Ctrl-Shift-Up Arrow	Moves the cursor from the implementation of a method to its class definition.
F5	Toggles a breakpoint for the current source code line (server applications only).
Ctrl-Tab	Selects the next source unit in the work area of the IDE, if one is present.
Ctrl-Shift-Tab	Selects the next source unit in the work area of the IDE, if one is present.
Ctrl-Shift-F	Displays the Find in Units dialog that allows you to perform text searches across all project source units, all open source units, or all source units in a specified path.
Ctrl-F	Shows the find toolbar at the bottom of the code editor. If the **Find Text at Cursor** option is enabled in the IDE, then the identifier at the current cursor position is automatically used as the search text. Please see the Modifying IDE Settings topic for more information on modifying the code editor search settings.

Ctrl-R	Shows the find and replace toolbar at the bottom of the code editor. If the **Find Text at Cursor** option is enabled in the IDE, then the identifier at the current cursor position is automatically used as the search text. Please see the Modifying IDE Settings topic for more information on modifying the code editor search settings.
Ctrl-G	Shows the goto line toolbar at the bottom of the code editor.

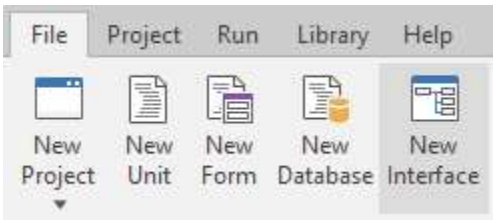
Context-Sensitive Help

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

3.17 Creating a New Control Interface

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

- Click on the **File** tab on the main menu.
- Click on the **New Interface** button on the File menu.

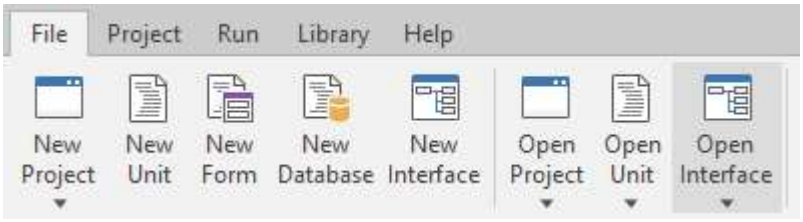


A new control interface will now appear in the Control Interface Designer.

3.18 Opening an Existing Control Interface

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

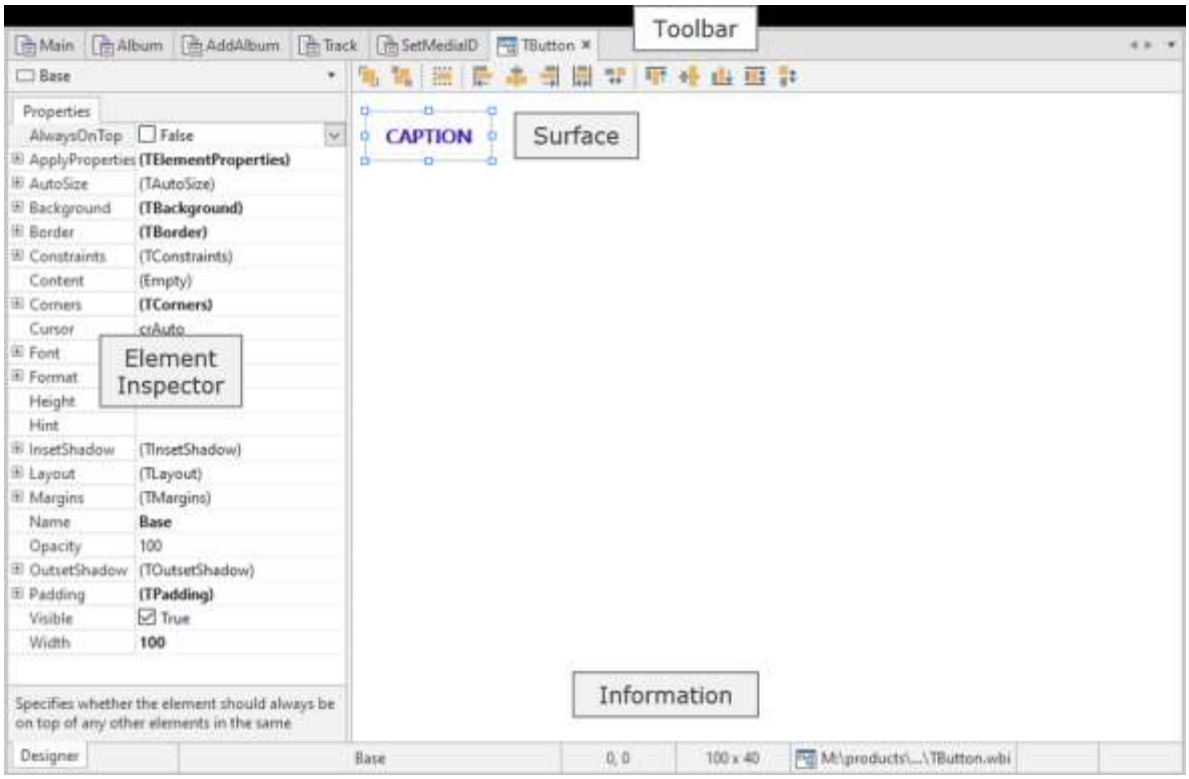
- Click on the **File** tab on the main menu.
- Click on the **Open Interface** button on the File menu.



A drop-down list of recently-opened control interfaces will appear beneath the Open Interface button. Select one of the recently-opened control interfaces by clicking on the desired control interface, or select another control interface by clicking on the **Open Interface from Folder** button at the bottom of the list of recently-opened control interfaces.

3.19 Using the Control Interface Designer

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



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

By default, the designer surface shows a grid to aid with element placement and alignment, and the grid guides (dots) are spaced apart at 10 pixel intervals. Please see the Modifying IDE Settings topic for more information on modifying the designer surface grid properties.

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

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

When a control interface designer is active in the IDE, the control interface navigator is accessible via the Tools window in the IDE:

The control interface navigator allows you to specify the control interface class name, add/rename/delete control interface states, and add/rename/delete interface elements for a given control interface state.

Specifying the Interface Class Name

By default, a control interface is assigned the base **TControl** control class name. You can change the control interface class name by selecting the root control interface class name node in the control interface navigator and pressing the **F2** key. You can then set the control interface class name to the desired name.

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

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

Adding a New Interface State

Control interfaces consist of one or more interface states. The default state is, by convention, named "Normal", and the first interface state that is created will automatically be named "Normal".

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

- Click on the Add State toolbar button:

- If the control interface class name node is selected when the Add State toolbar button is clicked, then the new interface state will only contain one base element and will be effectively empty. If a control interface state node is selected when the Add State toolbar button is clicked, then the new interface state will be a copy of the currently-selected control interface state. This makes it easy to create copies of existing states and then make minor modifications to reflect the state changes.

Renaming an Existing Interface State

Use the following steps to rename an existing interface state in the control interface:

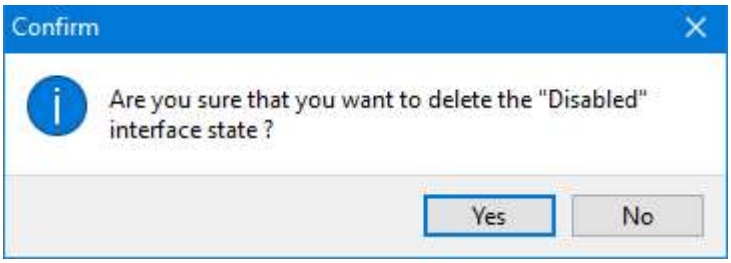
- Click on the desired state in the list of interface states in the control interface navigator.
- Press the **F2** key to start the rename, and then type in the new name of the interface state. Press the **Enter** key to complete the renaming.

Removing an Existing Interface State

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

- Click on the Remove State toolbar button:

- A dialog similar to the following will now appear:



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

Moving an Interface State

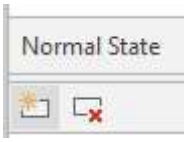
You can use drag and drop operations with the mouse to move an interface state to a different position in the list of defined interface states in the control interface navigator. Simply click on the desired interface state with the left mouse button, hold the left mouse button down, and drag the interface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

Adding a New Element to an Interface State

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

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

- Click on the Add Element toolbar button in the control interface navigator:



- The new element will be added as a child element of the currently-selected element in the control interface navigator.

Renaming an Existing Element in an Interface State

Use the following steps to rename an existing element in an interface state:

- Click on the desired element in the list of elements for the interface state in the control interface navigator.
- Press the **F2** key to start the rename, and then type in the new name of the element. Press the **Enter** key to complete the renaming.

Removing an Existing Element in an Interface State

Use the following steps to remove an existing element in an interface state:

- Click on the Remove Element toolbar button:

- The interface state element will then be removed.

Selecting an Element

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

> **Note**
> You can only select multiple elements within the same parent element.

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

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

Resizing an Element

Once an element has been placed on the designer surface, you will see that the element will have designer handles on all four sides and corners of the element:

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

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

Moving an Element

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

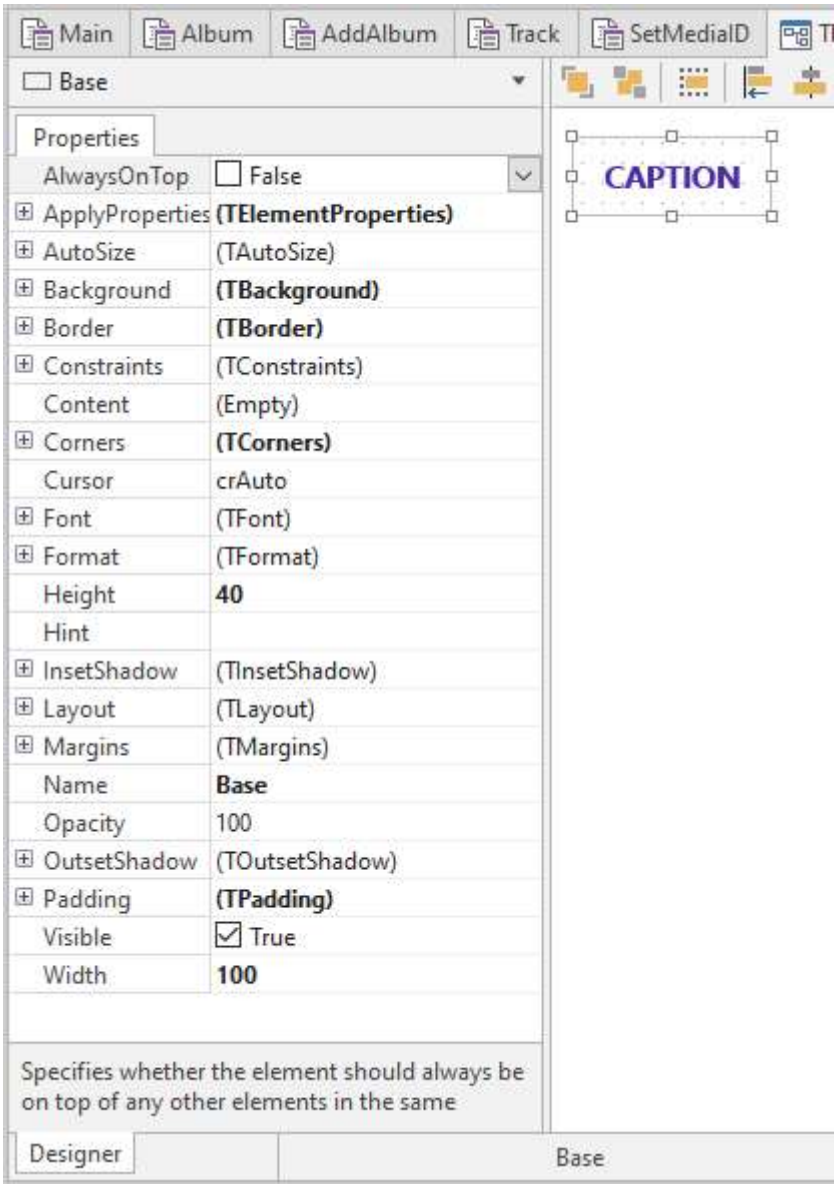
Element Layout and Alignment

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

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

Element Inspector

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



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

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

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

3.20 Viewing Messages

The messages area of the IDE provides compilation messages when building projects, deployment status information, and debugging information and messages. You can find out more information on sending debug messages to the messages area in the IDE in the Debugging topic.

Build Messages

There are three types of messages that may appear when an application is being built:

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

After an application build has completed, you will also see a messages summarizing the result of the build, including messages indicating which output files were emitted (and their location).

Deployment Messages

During deployment of an application, you will see various messages regarding the uploading of the application files in the messages area.

Design-Time Execution Messages

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

HTML Form Submittal Messages

If you use the special form submission URL (http://localhost/formsubmit or https://localhost/formsubmit) to submit an HTML form, the results of the submission will be echoed in the messages area. Double-clicking on the form submission results will display a debug dialog that will show you the complete set of form values received by the Internal web server built into the IDE. Please see the Using HTML Forms topic for more information.

3.21 Modifying IDE Settings

The Settings dialog allows you to configure the following aspects of the IDE:

- The project settings
- The code editor settings
- The code editor display settings
- The designer settings
- The component library settings
- The installed help files

The IDE settings are stored in the following file:

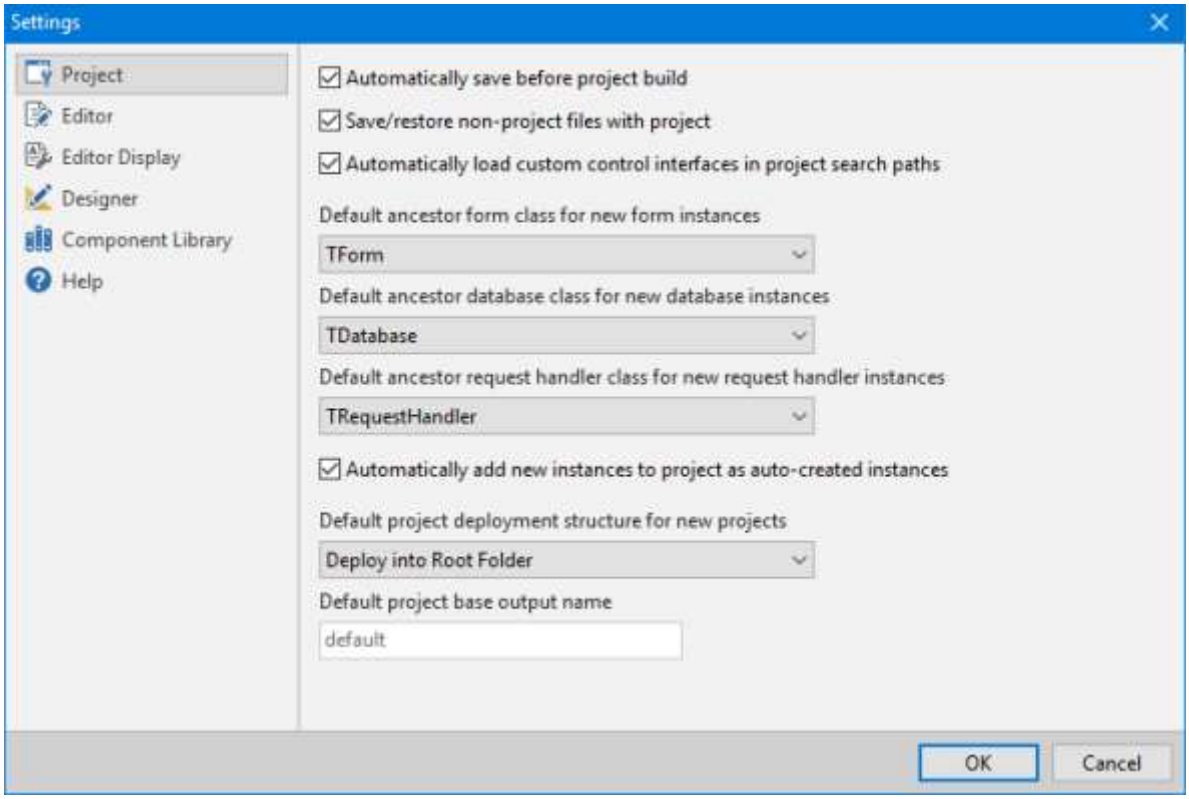
```
C:\Users\<UserName>\AppData\Local\Elevate Software\Elevate Web Builder
      3\ewbide.ini
```

where <UserName> is the name of the user account under which the IDE is being run.

Use the following steps to modify the settings for the IDE:

- Click on the **File** tab on the main menu.
- Click on the **Settings** button on the File menu to open the Settings dialog.

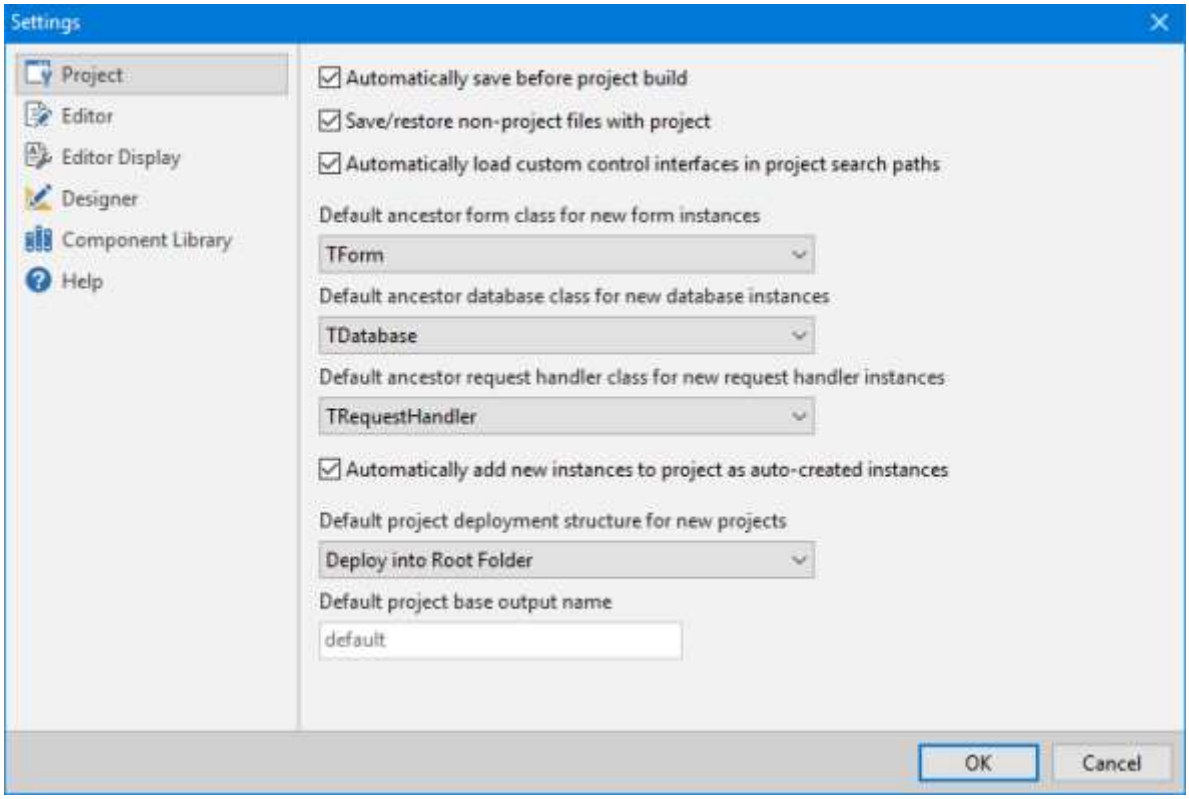
The Settings dialog will now appear:



After making any changes, click on the **OK** button to save the changes, or the **Cancel** button to discard the changes.

Project

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

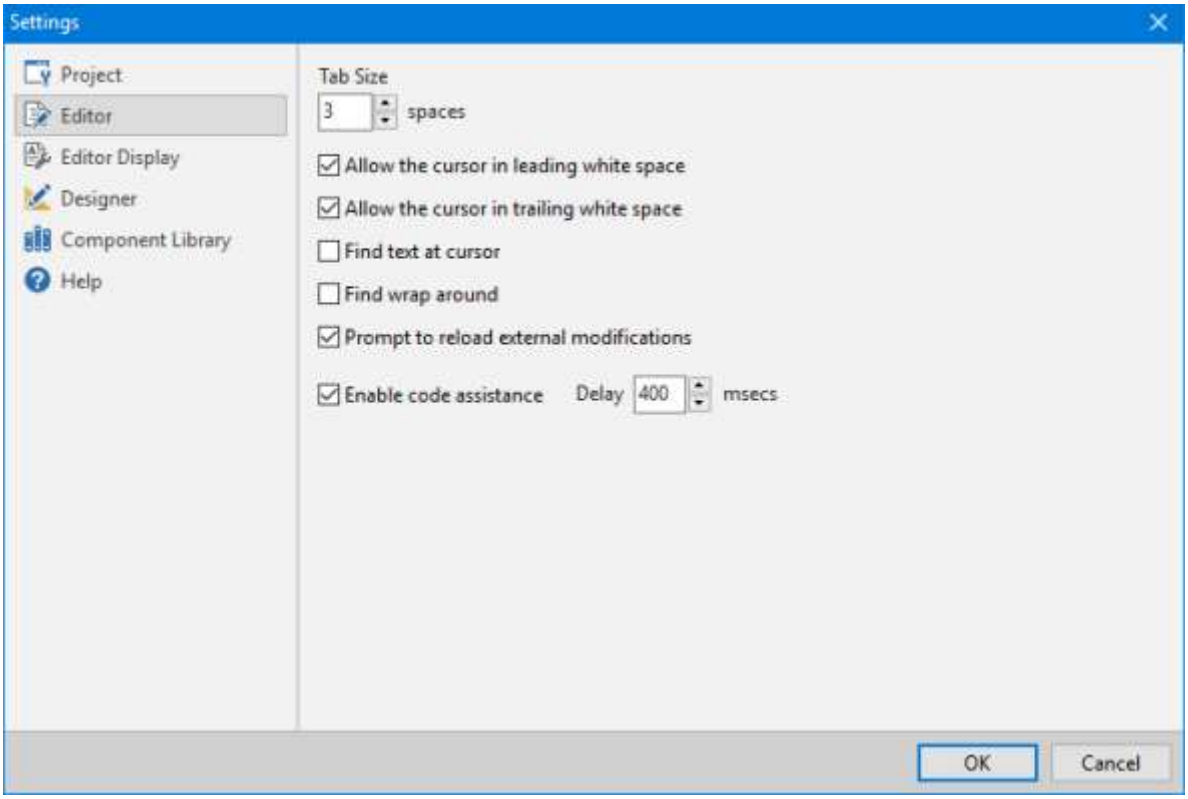


Option	Description
Automatically save before project build	Select this check box to make sure that the IDE automatically saves all modified units and project files before building the currently-loaded project. This option is selected by default.
Save/restore non-project files with project	Select this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

Default ancestor request handler class for new request handler instances	Specifies the default ancestor request handler class for the new request handler class selection dialog that is displayed when creating a new request handler in the IDE. The default ancestor request handler class is the TRequestHandler class.
Automatically add new instances to auto-created instances	Select this check box to make sure that any newly-created forms, databases, or request handlers are automatically added to the list of auto-created designer instances for the application. This option is selected by default.
Default project deployment structure for new projects	Use this combo box to select the default project deployment structure. This setting is only applicable to client applications, and the two options are to deploy the application files into the root folder (the default) or to deploy the application files into a project-specific folder. If you select the project-specific folder deployment structure, any new client projects will automatically have their project deployment path set to the same name as the project.
Default project base output name	If you select the project-specific folder deployment structure, you can then enter the default project base output name here. The default project base output name is used to construct the default output file names for both the HTML loader file and the JavaScript file when building and emitting client applications. The default value is "index".

Editor

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

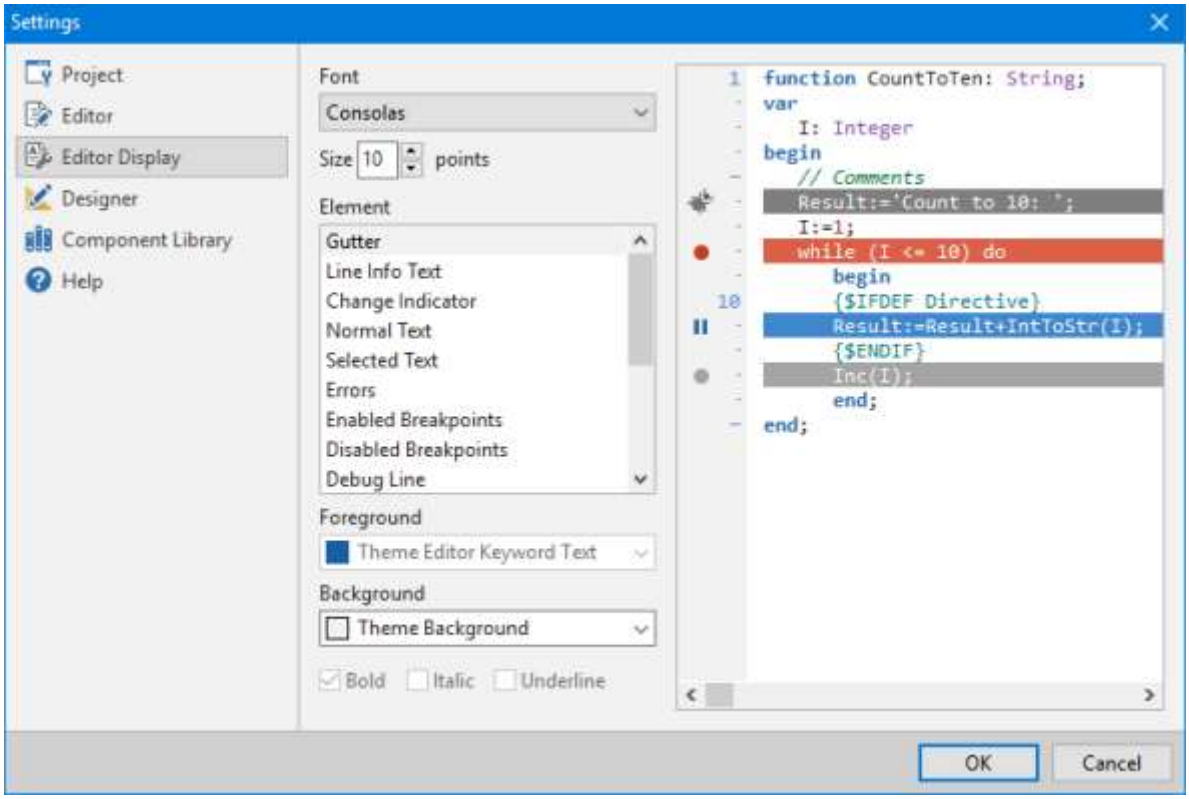


Option	Description

Tab Size	The number of spaces between each tab position. The default is 3 spaces.
Allow the cursor in leading white space	Select this check box in order to allow the cursor to be positioned in any leading white space in the code editor. By default, if you move to an area of the code editor that is leading white space, the cursor will be moved to the next closest source code to the white space. The definition of "white space" in this context is the area of the code editor where there is no source code present.
Allow the cursor in trailing white space	Select this check box in order to allow the cursor to be positioned in any trailing white space in the code editor. By default, if you move to an area of the code editor that is trailing white space, the cursor will be moved to the previous closest source code to the white space. The definition of "white space" in this context is the area of the code editor where there is no source code present.
Find text at cursor	Select this check box to have the code editor populate the Find or Replace search text box with the current word under the cursor when searching or replacing text in the code editor. By default, the last searched text will appear in the Find or Replace search text box.
Find wrap around	Select this check box to have the code editor wrap around to the start/end of the source when searching or replacing text in the code editor. The direction in which the searching or replacing wraps is determined by the direction of the search or replace operation. By default, the code editor will stop when reaching the start/end of the source during a search or replace operation.
Prompt to reload external modifications	Select this check box to have the code editor prompt the user when any source loaded in the code editor is modified by an external application. The prompt will ask the user to confirm whether they wish to load the modified source into the code editor. By default, the code editor will prompt the user when any source is changed by an external application.
Enable code assistance	Select this check box to enable code assistance in the code editor. Code assistance provides context-sensitive keyword and identifier hints after a short delay when you stop typing a keyword or identifier. The short delay is specified via the Delay edit box. Code assistance is enabled, by default, and the default delay is 400 millseconds.

Editor Display

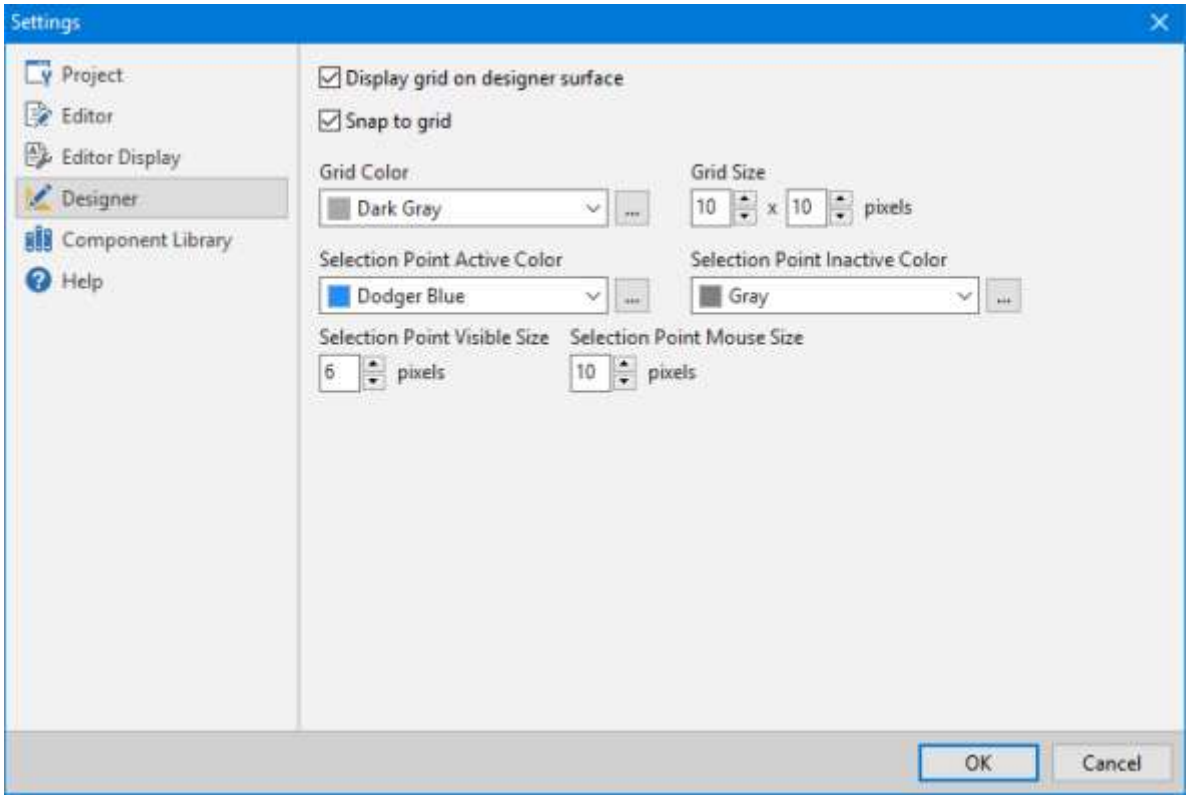
The Editor Display page provides options for modifying the code editor settings.



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

Designer

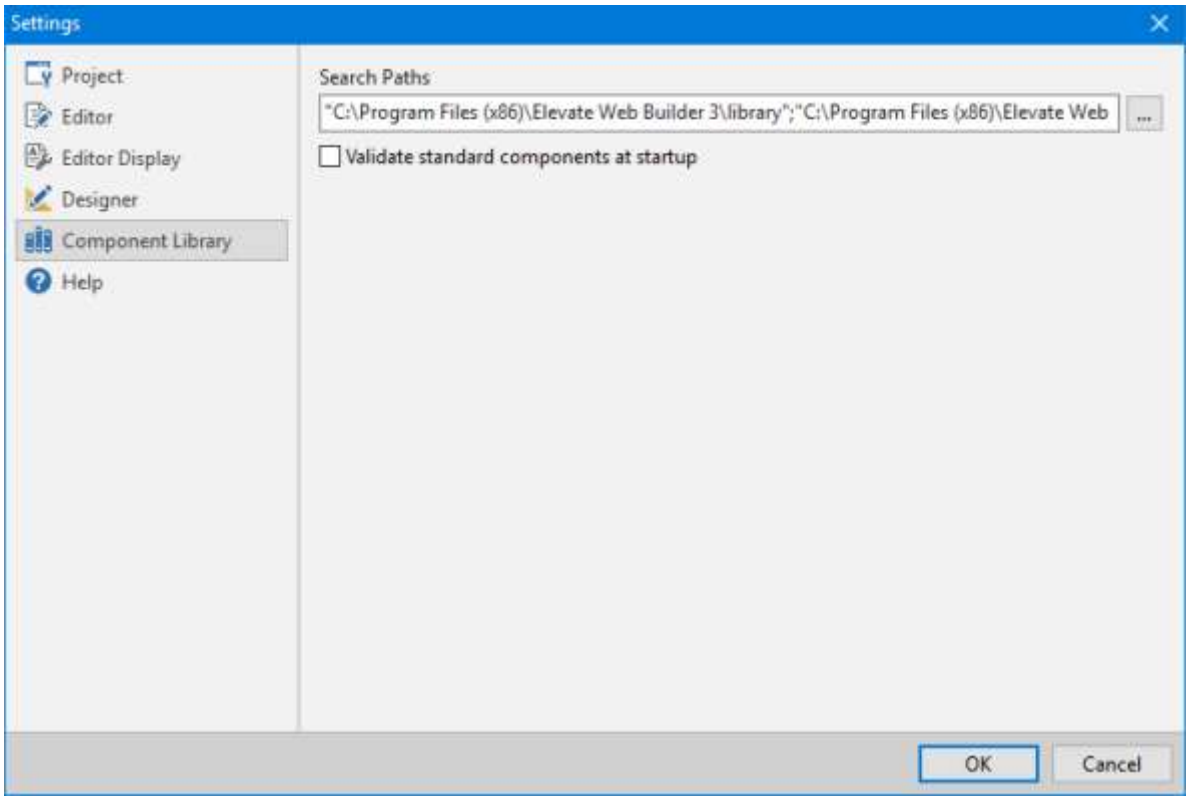
The Designer page provides options for modifying the designer settings.



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

Component Library

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

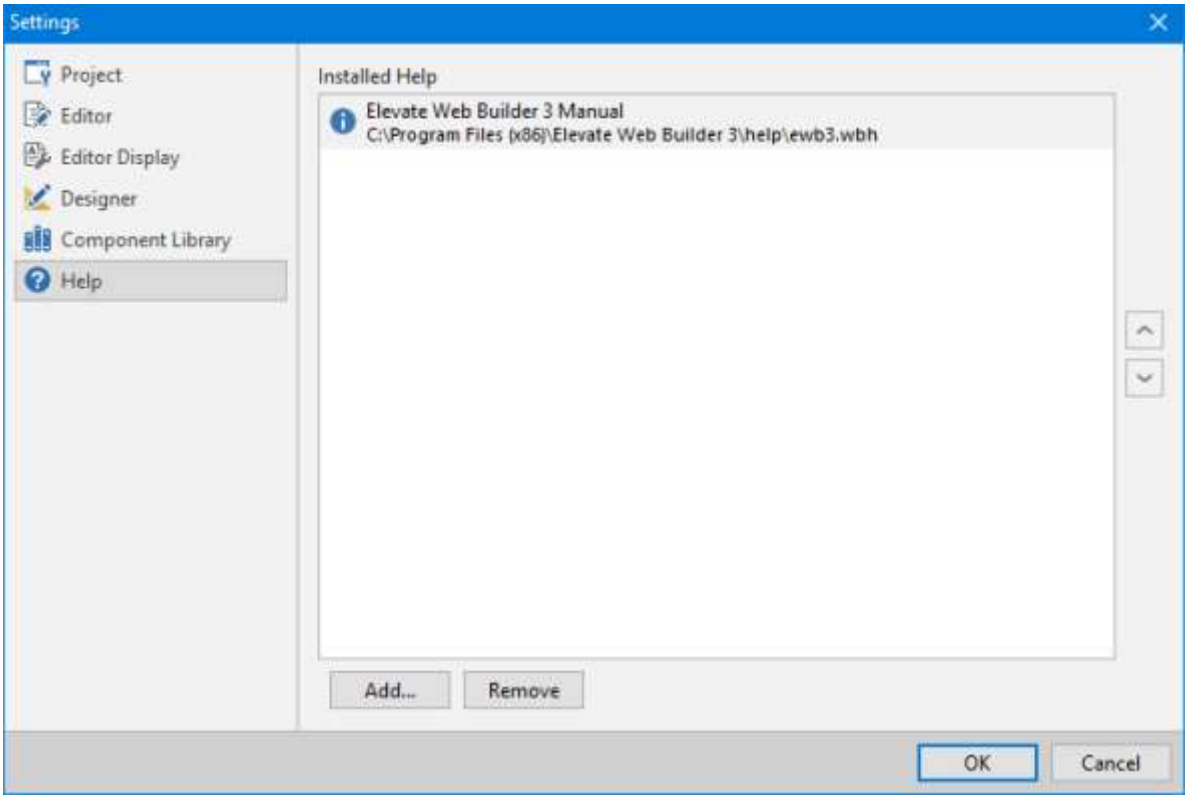


Option	Description
Search Paths	The component library search paths are used to specify where the component library source unit files are located. These search paths ensure that the compiler can always find the component units (and any referenced control interfaces) installed into the component library,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

Help

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

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

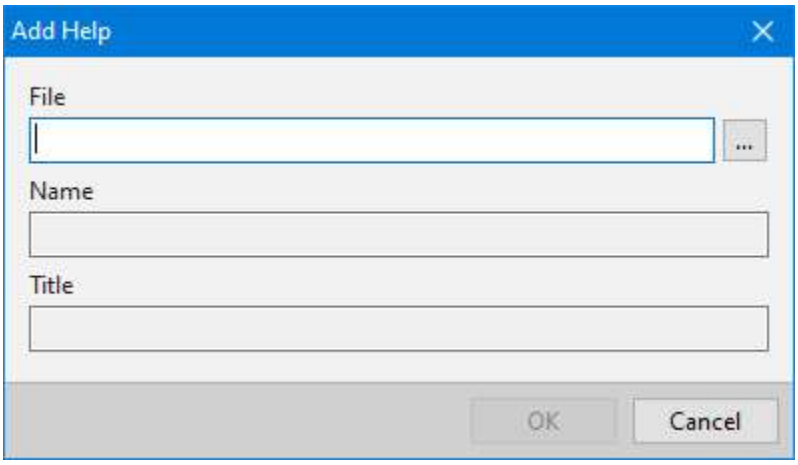


Adding Help

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

- Click on the **Add** button.

- The **Add Help** dialog will appear.



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

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

Removing Help

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

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

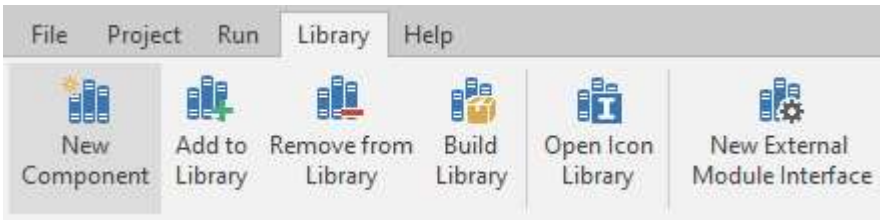
- Click on the **Remove** button.

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

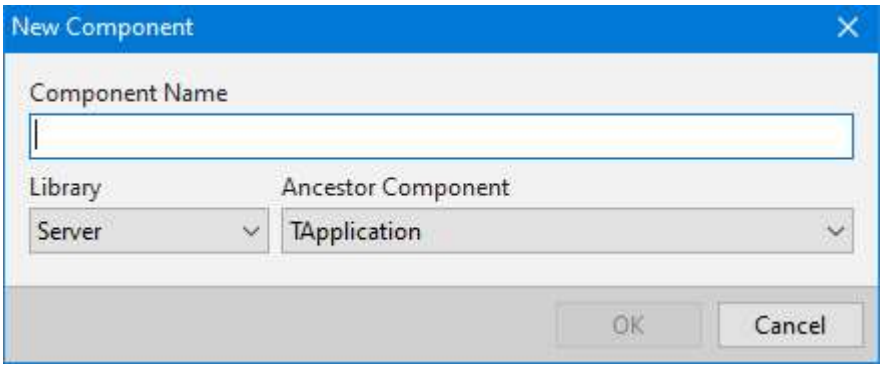
3.22 Creating a New Component

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

- Click on the **Library** tab on the main menu.
- Click on the **New Component** button on the Library menu:



- The New Component dialog will now appear:



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

  Next, select the applicable component library for the ancestor component class so that you can select the proper ancestor class name.

  > **Note**
  > If a project is open in the IDE, then the default selected component library will be the component library used by the open project. If a project is not open in the IDE, then the default selected component library will be the last component library used for any operation in the IDE, with the initial component library being the **Client** component library.

  Finally, select the ancestor component class name for the new component.

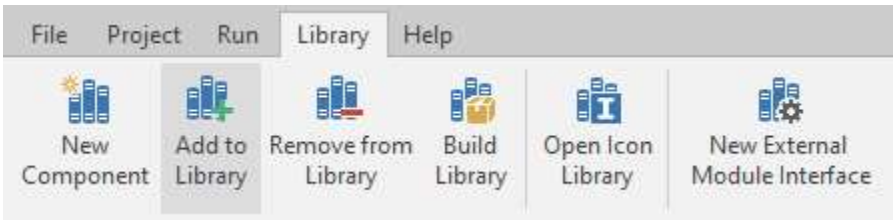
- Click on the **OK button**.

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

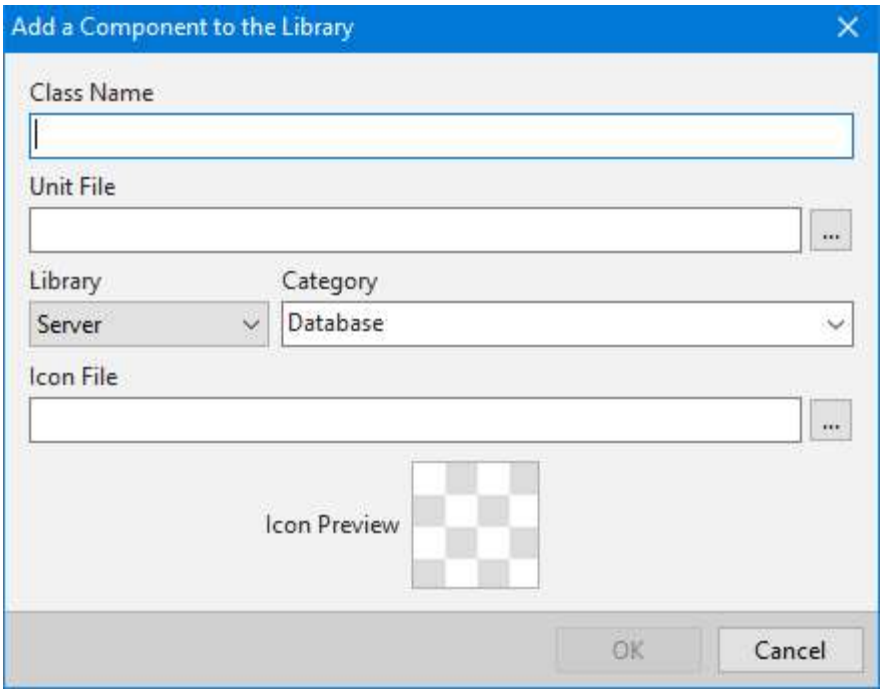
3.23 Adding a Component to the Component Library

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

- Click on the **Library** tab on the main menu.
- Click on the **Add to Library** button on the Library menu:



- The Add Component dialog will now appear:



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

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

  Next, select which component library you would like to add the component to.

  > **Note**
  > If a project is open in the IDE, then the default selected component library will be the component library used by the open project. If a project is not open in the IDE, then the default selected component library will be the last component library used for any operation in the IDE, with the initial component library being the **Client** component library.

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

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

  The IDE supports high-DPI screen resolutions, so you can include other variations of the icon file in the same directory as the default icon file using the following file pattern:

  ```
  <IconFileNameRoot><PixelSize>.<IconFileNameExt>
  ```

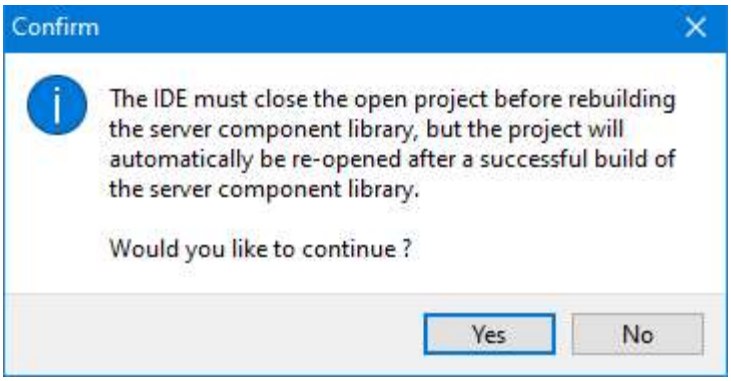
  where <PixelSize> can be:

Pixel Size	Screen Resolution
20	A 20 by 20 pixel icon file for 120-DPI screen resolutions
24	A 24 by 24 pixel icon file for 144-DPI screen resolutions
32	A 32 by 32 pixel icon file for 168-DPI or higher screen resolutions

  > **Note**
  > The default 16 by 16 pixel icon file should not include the pixel size as part of the file name. Also, this step is optional, if you don't specify an icon file, or if the specified icon file is invalid, the IDE will use a default, generic icon for the component in the component library.

- Click on the **OK** button.

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

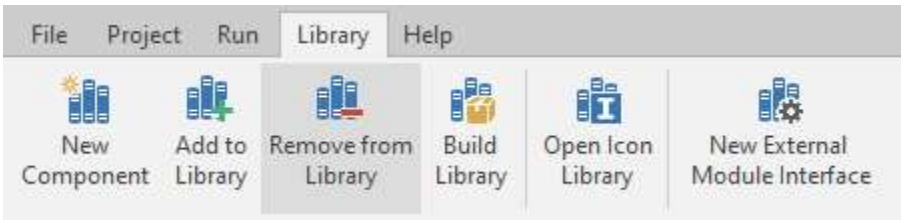


Click on the **Yes** button to continue with rebuilding the component library.
The component library will now automatically be rebuilt and, if there were no errors during the compilation of the component library, the component just added will now appear in the component library in the specified category.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

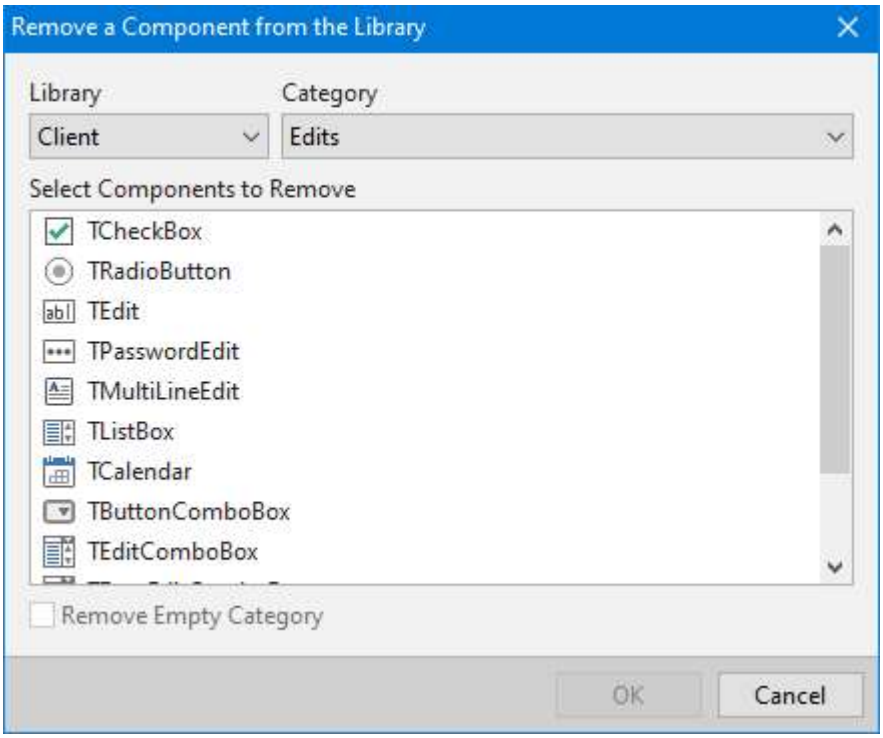
3.24 Removing a Component from the Component Library

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

- Click on the **Library** tab on the main menu.
- Click on the **Remove from Library** button on the Library menu:



- The Remove Component dialog will now appear:



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

  Note
  If a project is open in the IDE, then the default selected component library will be the component library used by the open project. If a project is not open in the IDE, then the default selected component library will be the last component library used for any operation in the IDE, with the initial component library being the **Client** component library.

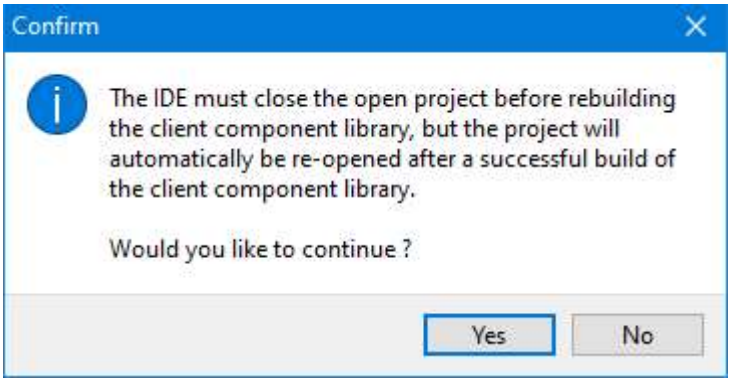
  Next, select the category where the component that you wish to remove is installed.

  Next, select any or all component(s) that you wish to remove from the component library. You can select multiple components by holding down the Ctrl key while clicking on the components, or holding down the Shift key while clicking on or navigating the components with the arrow keys.

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

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

- Click on the **OK** button.
- If there is a project open in the IDE, then you will see the following dialog appear:



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

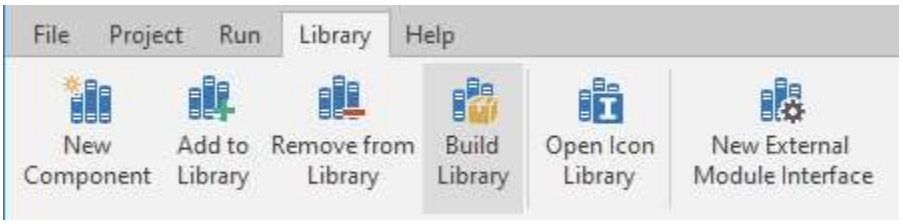
The component library will now automatically be rebuilt and, if there were no errors during the compilation of the component library, the selected component(s) will be removed from the selected category in the component library. If there were one or more errors during the compilation of the component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

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

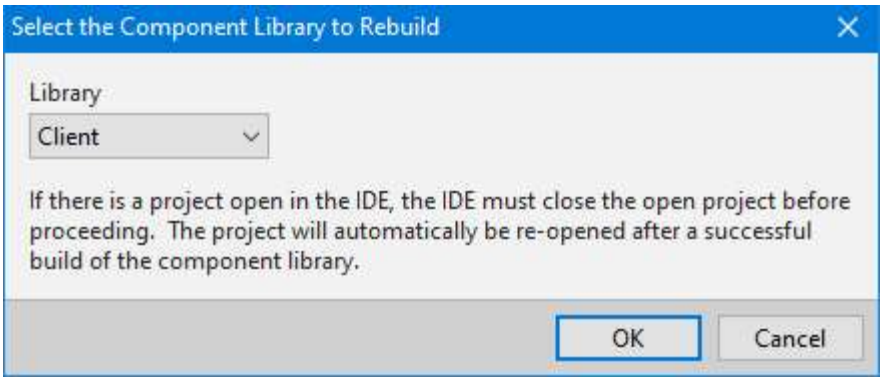
3.25 Building the Component Library

Use the following steps to build the component library:

- Click on the **Library** tab on the main menu.
- Click on the **Build Library** button on the Library menu:



- The following dialog will now appear:



- In this dialog, select the component library that you wish to build.

> **Note**
> If a project is open in the IDE, then the default selected component library will be the component library used by the open project. If a project is not open in the IDE, then the default selected component library will be the last component library used for any operation in the IDE, with the initial component library being the **Client** component library.

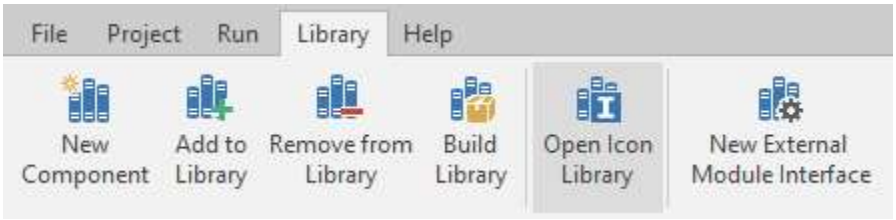
- Click on the **Yes** button to continue with building the component library.

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

3.26 Opening the Icon Library

Use the following steps to open the icon library:

- Click on the **Library** tab on the main menu.
- Click on the **Open Icon Library** option on the Library menu:



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

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

If you want to customize the icon library for the active project, simply use the **File/Save As** menu option to save the default icon library interface file in a different folder/directory.

> **Warning**
> You should always use the default interface file name "TIconLibrary.wbi", even for customized icon libraries. If you do not use the correct interface file name, then the customized icon library will be ignored.

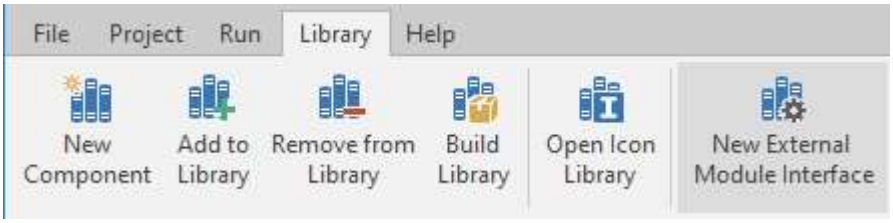
3.27 Creating a New External Module Interface Unit

External modules are special native libraries created with Delphi XE2 or higher that expose native classes, types, functions, procedures, and/or global variables to the server application runtime in the Elevate Web Builder Web Server. Once you have created and built an external module using Embaracadero RAD Studio, you will need to create an interface unit for the external module so that the compiler knows how to resolve the native classes, types, functions, procedures, and/or global variables exposed in the external module. Once you have created an external module interface unit, you can use the functionality in the external module just like any other code by simply including a reference to the external module interface unit.

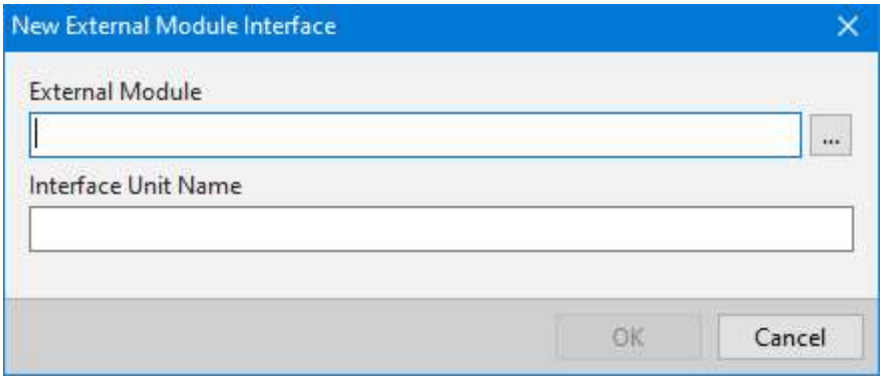
> **Note**
> Delphi does not support getting information about default values for the parameters of methods, functions, or procedures, so this information is **not** included in the generated external module source unit.

Use the following steps to create a new external module interface unit:

- Click on the **Library** tab on the main menu.
- Click on the **New External Module Interface** option on the Library menu:



- The New External Module Interface dialog will now appear:



- In this dialog, specify the file name of the external module that you wish to generate the interface unit for. You can type in the file name directly, or use the browse button (...) to select the external module using a common Windows file dialog.

  Next, specify the name you would like to give the external module interface unit.

- Click on the **OK button**.

A new source unit containing the external module's exposed classes, types, functions, procedures, and/or global variables will now appear in the code editor. Please see the Using the Code Editor topic for more information on using the code editor.

Chapter 4
Using Visual Controls

4.1 Standard Controls

The following are the visual controls in the standard component library. They are grouped and ordered by the category in which they are installed and displayed in the component library in the IDE. Most of the visual controls are capable of being bound to a dataset. Please see the Binding Controls to DataSets topic for more information.

Labels

The label controls are used for displaying text in various ways.

Control	Description
TLabel	Label control
THTMLLabel	HTML label control
TBalloonLabel	Balloon label control
TAlertLabel	Alert label control

Buttons

The button controls are used for taking an action when the control is clicked and can use both text and icons to convey the type of action.

Control	Description
TButton	Button control
TDialogButton	Dialog button control
TIconButton	Icon button control

Edits

Edit controls are used to display and edit data in various ways.

Control	Description

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

Grids

Grid controls are used to display and edit data inline in a tabular form.

Control	Description
TGrid	Grid control

Containers

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

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

Graphics

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

Control	Description

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

Indicators

Indicator controls show progress and other types of graphic information:

Control	Description
TProgressBar	Progress bar control

Multimedia

Multimedia controls allow the playback of both audio and video:

Control	Description
TAudio	Audio playback control
TVideo	Video playback control

Menus

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

Control	Description
TMenu	Menu control
TMenuBar	Menu bar control

ToolBars

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

Control	Description
TToolBar	Toolbar control
TDataSetToolBar	Dataset toolbar control

Browser

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

Control	Description

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

4.2 Creating and Showing Forms

The following are the forms and dialogs in the standard component library.

Form	Description
 TForm	Standard scrollable form control
 TDialog	Non-scrollable dialog control with modal functionality
 TFrame	Lightweight, non-scrollable frame control

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

Creating a Form at Design-Time

The easiest way to create a form is by using the IDE to create a new form. When a form is created at design-time in the IDE, it is automatically designated as an auto-create form in the application project, which means that the form will automatically be created at application startup and be owned by the global TApplication Application instance. When a form or any TComponent-descendant component instance is owned, the instance is automatically freed when the owning component is freed. Please see the Visual Client Applications topic for more information on the global Application instance.

The TApplication Run method accepts an optional string parameter that indicates the name of the form that should be treated as the main form for the application. The main form of the application is automatically shown and brought to the front at application startup. If this parameter is not specified or is not a valid form name, then the first form in the list of auto-create forms is considered the main form of the application. For example, the following shows the main program source of an application that has one auto-create form:

```
project MusicCollection;

contains Main, Album, AddAlbum, Track, SetMediaID;

uses WebForms, WebCtrls;

begin
   Application.CreateForm(TMainForm);
   Application.Run('MainForm');
end.
```

In this case the form whose Name property is equal to the parameter passed to the Run method will be considered the main form. If no form exists with the name of "MainForm", then the TMainForm form class instance will be considered the main form of the application and will automatically be shown when the Application Run method is executed.

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

Creating a Form at Run-Time

You can also create a form instance at run-time using code. This is useful for forms that are not used very often and for which having them auto-created would be a waste of memory. The following is an example of creating a form and showing it (modally) at run-time. The EditFolderDlg variable is declared as a global variable in the interface section of the TEditFolderDlg's unit (EditFolder) and is automatically managed by the design-time environment in the IDE.

```
uses EditFolder;

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

Note
The Owner parameter for the Create constructor method is set to nil, which indicates that the memory associated with the form instance will be manually managed and not owned by another component. As noted above, when a form or any TComponent-descendant component instance is owned, the instance is automatically freed when the owning component is freed.

Showing and Hiding a Form

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

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

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

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

Modal Forms

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

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

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

```
uses ProgFrm;

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

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

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

```
uses ProgFrm;

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

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

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

Form Events

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

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

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

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

4.3 Showing Message Dialogs

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

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

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

```
procedure ShowTrialMessage;
begin
   if IsTrialVersion and (TrialDaysLeft > 0) then
      ShowMessage('You are using a trial version and have '+
                  IntToStr(TrialDaysLeft)' evaluation days '+
                  'left until your trial version expires.')
   else
      ShowMessage('Your trial version has unfortunately expired.');
end;
```

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

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

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

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

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

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

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

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

4.4 Showing Progress Dialogs

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

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

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

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

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

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

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

4.5 Using HTML Forms

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

Input Controls

Any control that descends from the base TInputControl control class can be used as an input control for an HTML form. The Name property of the control instance is used as the field name within the HTML form and the field value is assigned internally via each control instance.

The following standard controls can be used as input controls in HTML forms:

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

Submitting the Form

There are two ways to submit an HTML form to the web server:

- By using the THTMLForm instance's Submit method.
- By assigning the THTMLForm instance to a TServerRequest instance using the TServerRequest Assign method and then executing the request. This technique is sometimes preferable with file uploads because it allows you to track the progress of the upload using a TServerRequest OnProgress event handler, and it prevents the web browser from navigating away from the client application in response to the HTML form submission.

Submitting the Form Using the THTMLForm Submit Method

In order to submit the input information as an HTTP POST request to the web server using the THTMLForm Submit method, complete the following steps:

- Make sure that the THTMLForm instance's Encoding property is set to feMultiPartFormData. You can use other encoding types, but this is the default and supports the most common type of form submission, including submitting files using the TFileComboBox control.
- Make sure that the THTMLForm instance's Method property is set to fmPost. This is the default value, so you'll probably never need to change this property.
- Make sure that the THTMLForm instance's URL property is set to the desired URL.
- Call the THTMLForm instance's Submit method to perform the HTML form submission.

Submitting the Form Using a TServerRequest

In order to submit the input information as an HTTP POST request to the web server using a TServerRequest component, complete the following steps:

- Make sure that the THTMLForm instance's Method property is set to fmPost. This is the default value, so you'll probably never need to change this property.
- Make sure that the THTMLForm instance's URL property is set to the desired URL.
- Call the TServerRequest instance's Assign method to assign the HTML form to the server request. This method will assign the THTMLForm Method and URL properties to the corresponding TServerRequest Method and URL properties, as well as all of the form data. This form data will stay with the TServerRequest instance until the TServerRequest Reset method is called.

> **Note**
> There is no need to set the **Content-Type** header using the TServerRequest Headers property when submitting HTML forms using the TServerRequest component. It will be automatically set to "multipart/form-data" by the web browser when the request is executed.

- Call the TServerRequest instance's Execute method to execute the request and perform the HTML form submission.

Testing Form Submissions

The internal web server embedded in the IDE includes support for echoing back any values submitted as part of an HTML form submission. Just be sure to use the following URL for the THTMLForm instance's URL property:

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

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

You can also test form submissions with a secure connection using the following URL:

```
https://localhost/formsubmit
```

Note
The above URL assumes that the internal web server is listening on the standard secure port 443. Please see the Modifying Environment Options topic for more information on configuring the internal web server.

Redirecting Form Submission Output

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

Note
You cannot redirect the output of an HTML form submission that is completed using a TServerRequest instance instead of the THTMLForm Submit method. In addition, form submissions completed using a TServerRequest instance will never cause the web browser to navigate away from the client application.

4.6 Layout Management

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

Control Layout Properties

Each control in a visual client application possesses several key properties that control the layout of the control:

Left, Top, Width, and Height

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

Layout Order

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

Layout

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

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

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

Constraints

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

Margins

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

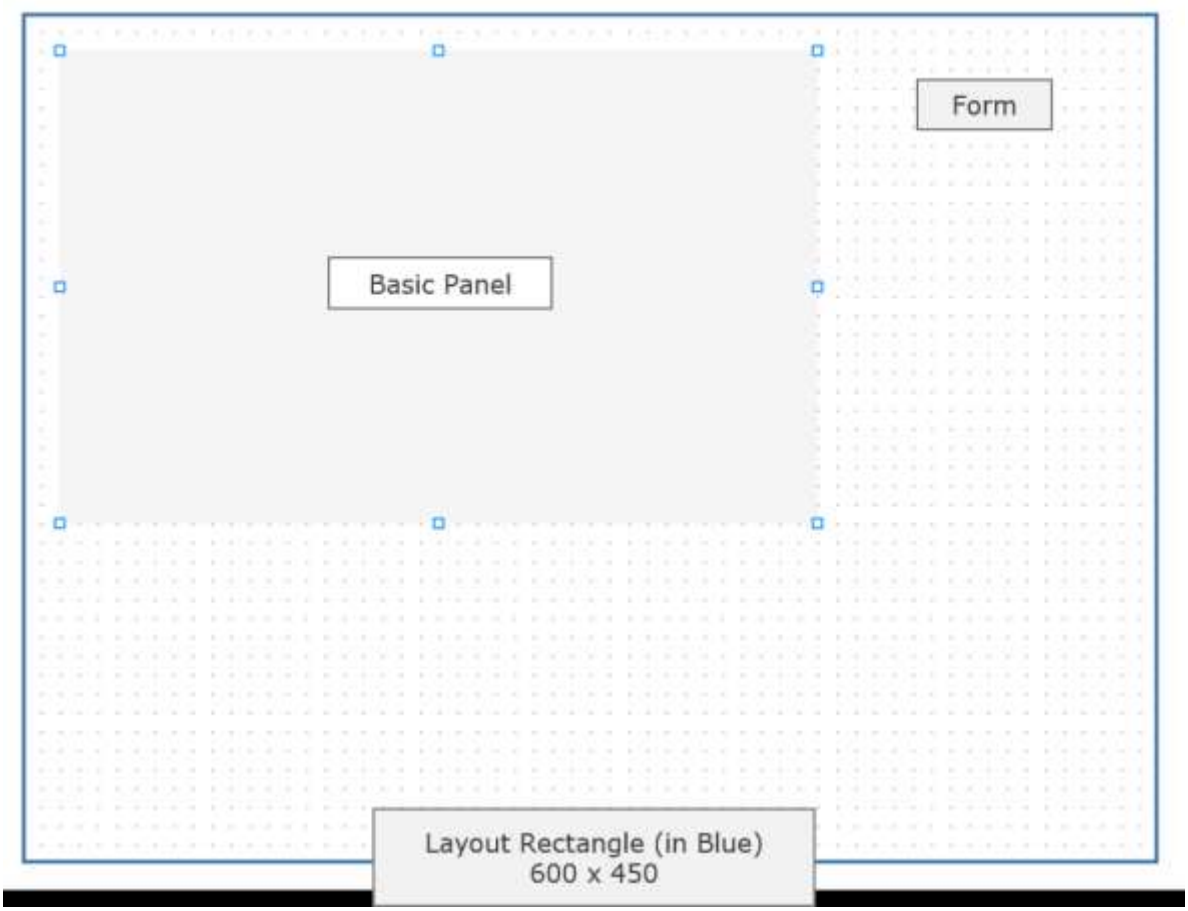
Padding and Borders

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

Layout Rectangle

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

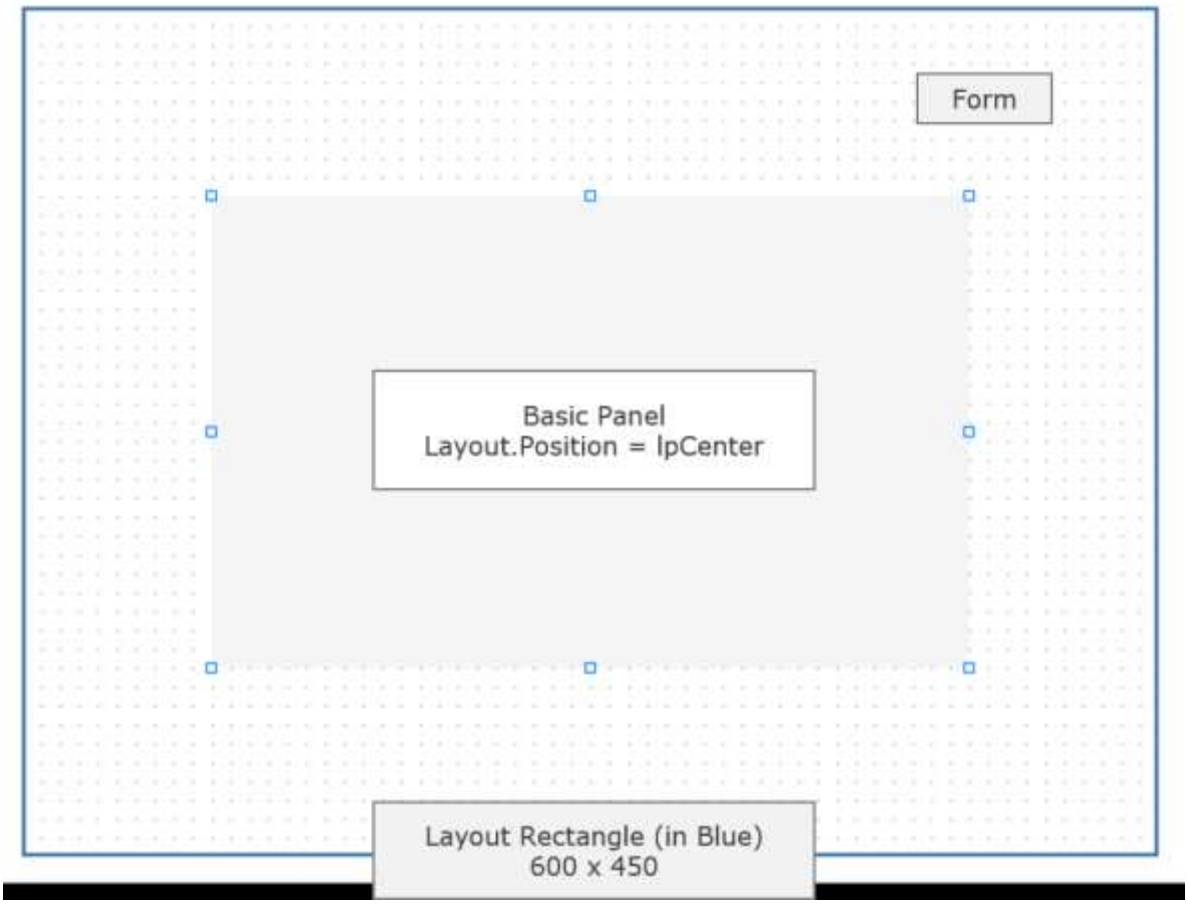
To illustrate the concept of the layout rectangle in its most basic form, let's place a single TBasicPanel control instance on a form (TForm-descendant instance). In this case, the form instance is the container control and the TBasicPanel instance is the child control. Because the form instance does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



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

DisplayOrder	1
Height	250
Hint	
⊞ InsetShadow	(TInsetShadow)
⊟ Layout	(TLayout)
Consumption	lcNone
Overflow	loNone
Position	lpCenter
Reset	☐ False
Stretch	lsNone
LayoutOrder	0
Left	100
⊞ Margins	(TMargins)

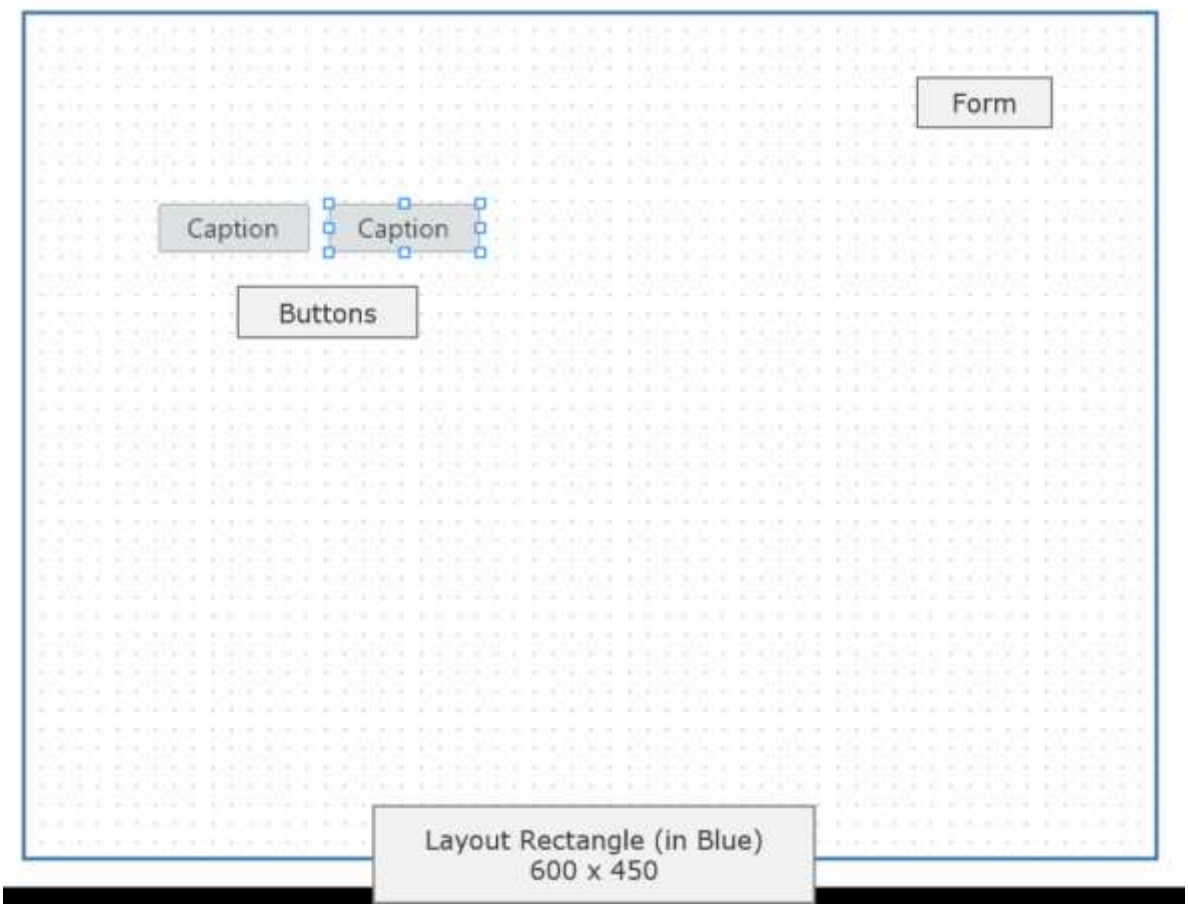
The resulting layout looks like this:



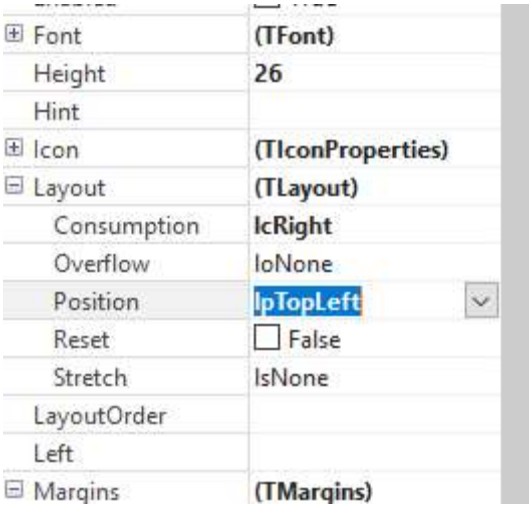
As you can see, the layout functionality used the layout rectangle of the form instance to center the defined dimensions of the TBasicPanel control instance. In this case, the layout rectangle was used for positioning only.

Consuming Space in the Layout Rectangle

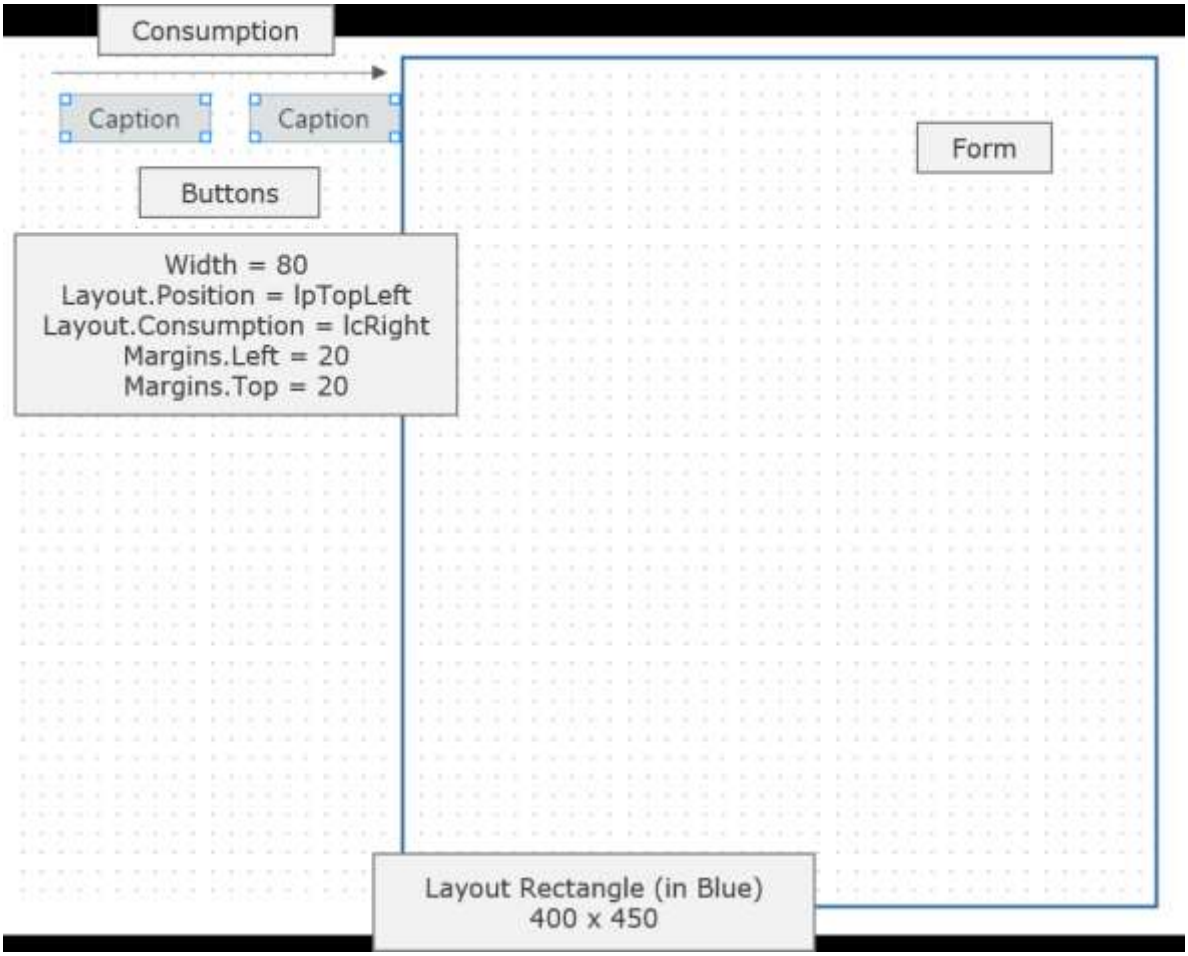
To illustrate how space consumption affects the layout rectangle, let's place two TButton control instances on a form (default width of 80 pixels). Again, the form instance is the container control and the TButton instances are the child controls, and the initial layout rectangle is the same as the client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



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

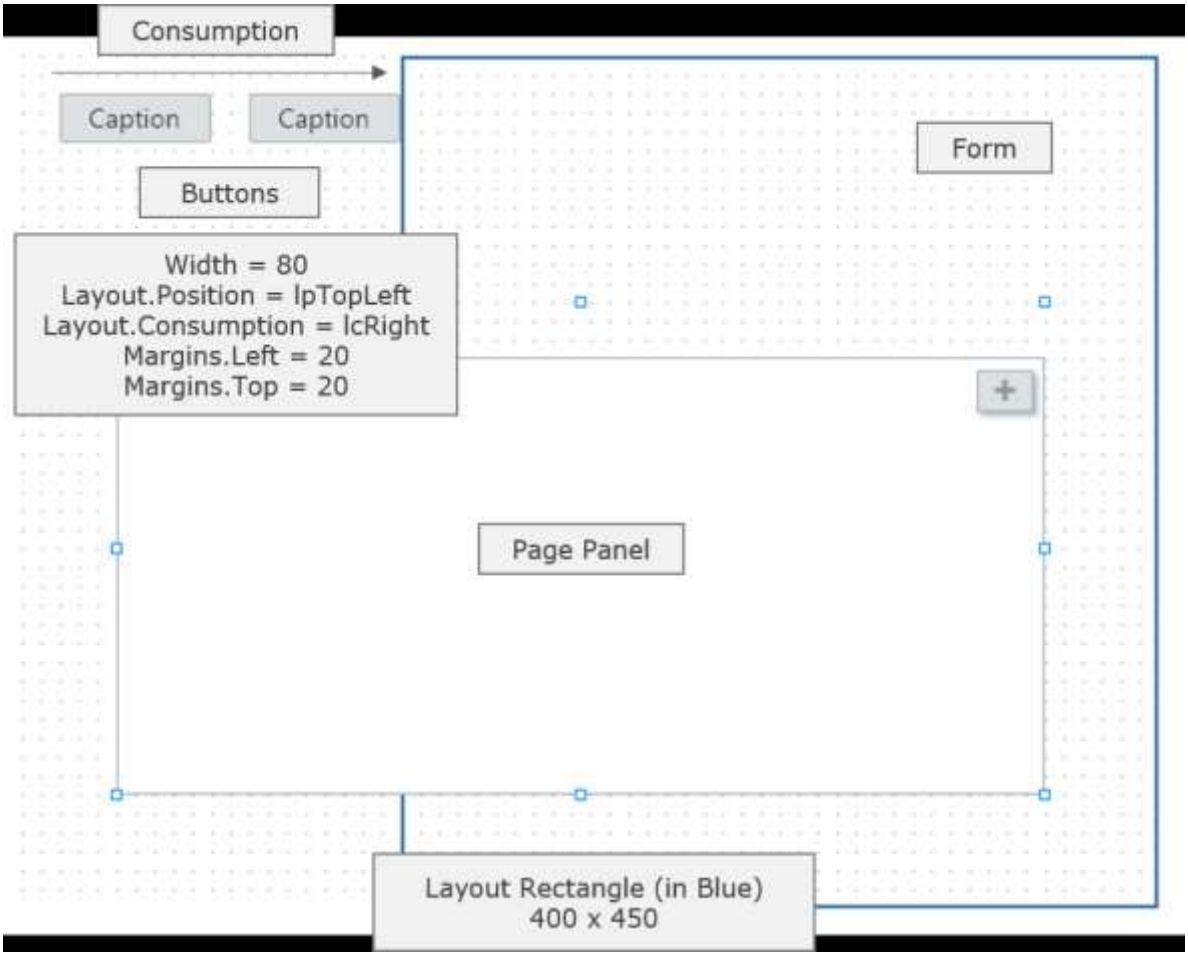


The resulting layout looks like this:



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

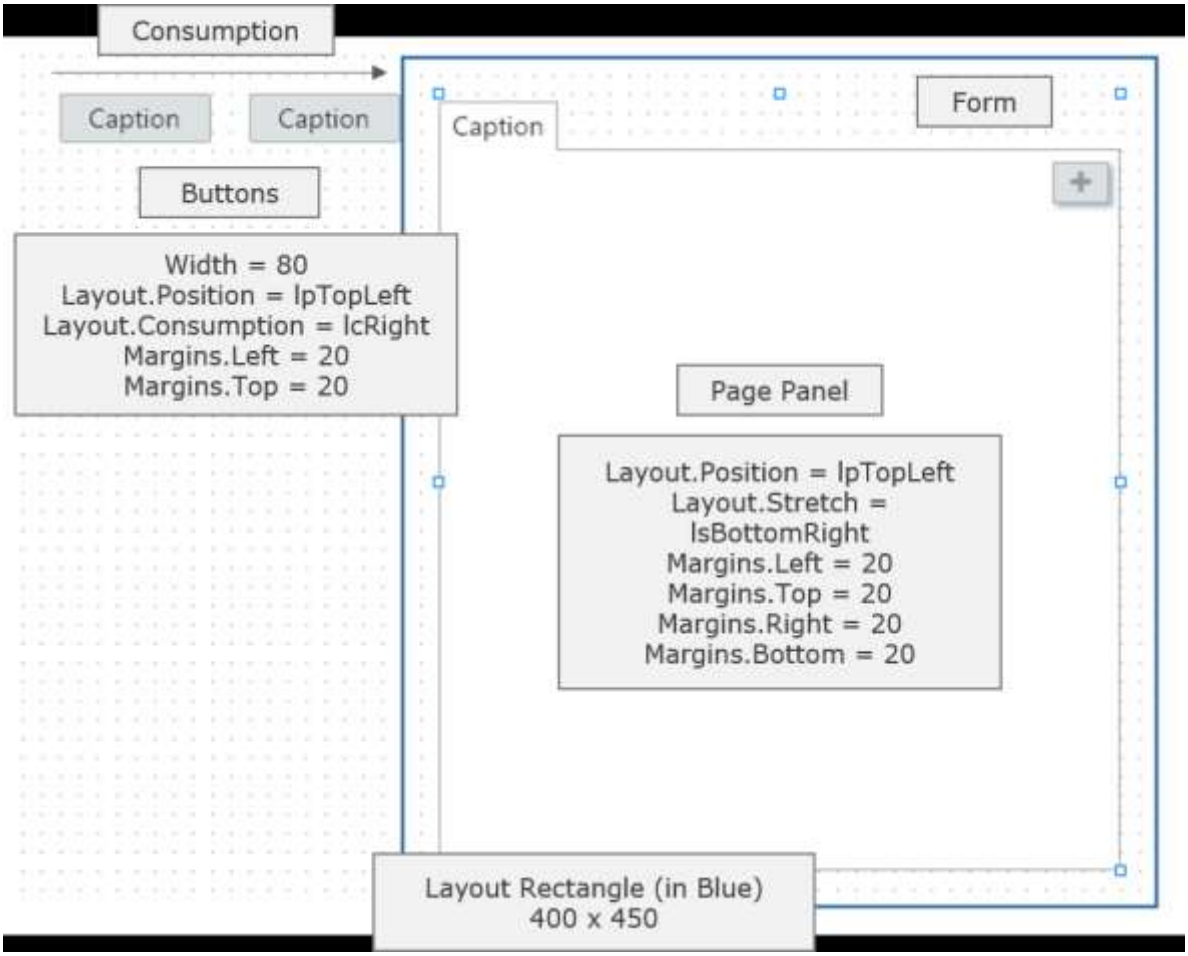
In most cases a form would not consist of just two buttons, so let's continue with the layout by placing a TPagePanel control instance on the form.



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

⊟ Layout	**(TLayout)**
Consumption	lcNone
Overflow	loNone
Position	**lpTopLeft**
Reset	☐ False
Stretch	**lsBottomRight**
LayoutOrder	**3**
Left	**220**
⊞ Margins	**(TMargins)**
Name	**PagePanel1**
Opacity	100
⊞ Padding	(TPadding)
PageNavigation	☐ False

The resulting layout looks like this:



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

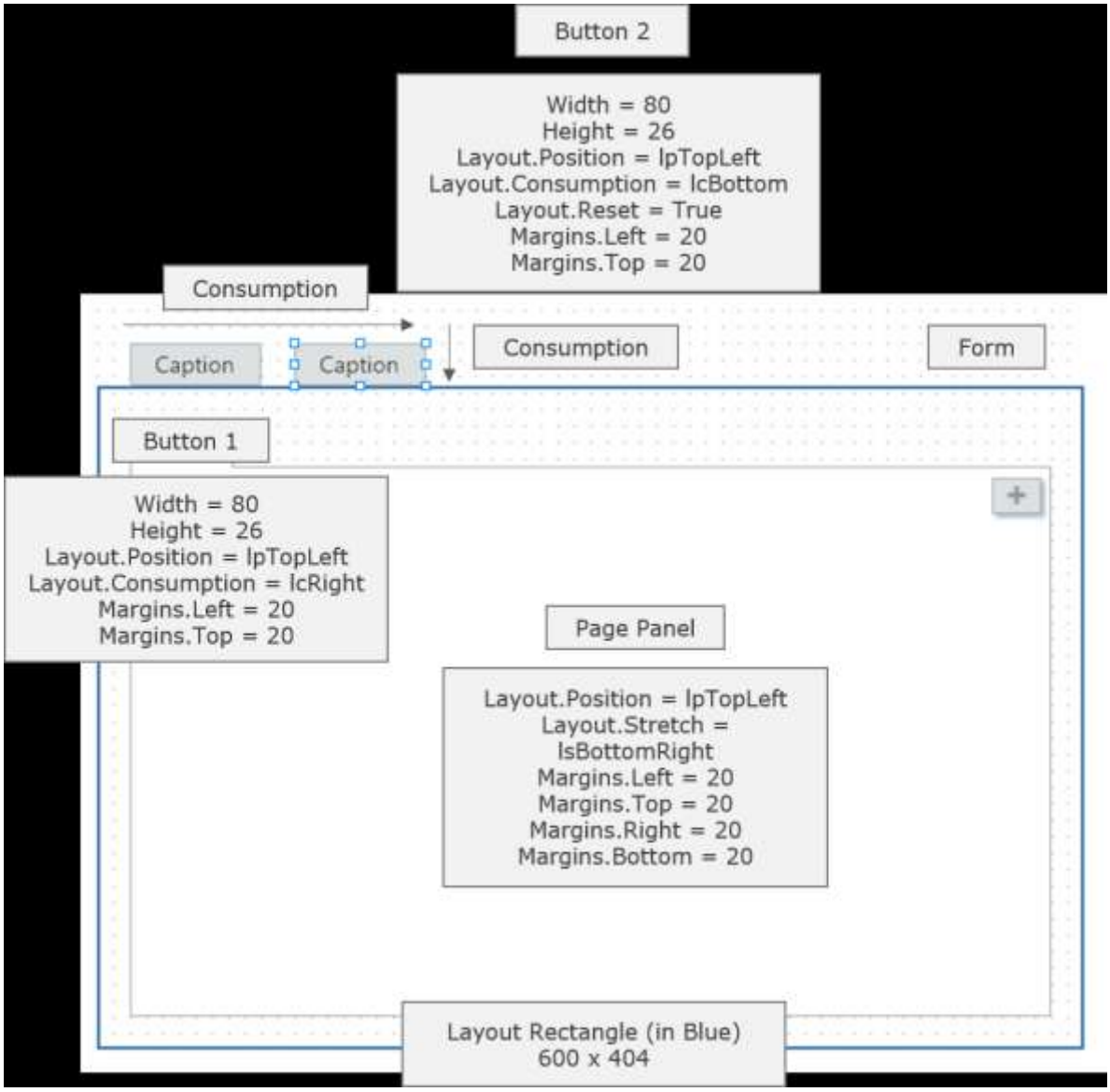
To fix this, we only need to change two properties for the second TButton instance: we need to specify that the Layout.Consumption property should be **lcBottom** and that the Layout.Reset button should be **True**:

⊞ Font	**(TFont)**
Height	**26**
Hint	
⊞ Icon	**(TIconProperties)**
⊟ Layout	**(TLayout)**
Consumption	**lcBottom**
Overflow	loNone
Position	**lpTopLeft**
Reset	☑ **True**
Stretch	lsNone
LayoutOrder	**1**
Left	**120**
⊞ Margins	**(TMargins)**

Changing these two properties in this manner does two things:

- It changes the consumption direction towards the bottom of the layout rectangle, which is where we want the TPagePanel instance to be placed.
- It resets the layout rectangle back to the last reset point. Since this is the only control whose Layout.Reset property is set to True, this means that the last reset point is the original layout rectangle for the form. The reset of the layout rectangle will occur **before** the control consumes any space.

The resulting layout looks like this:



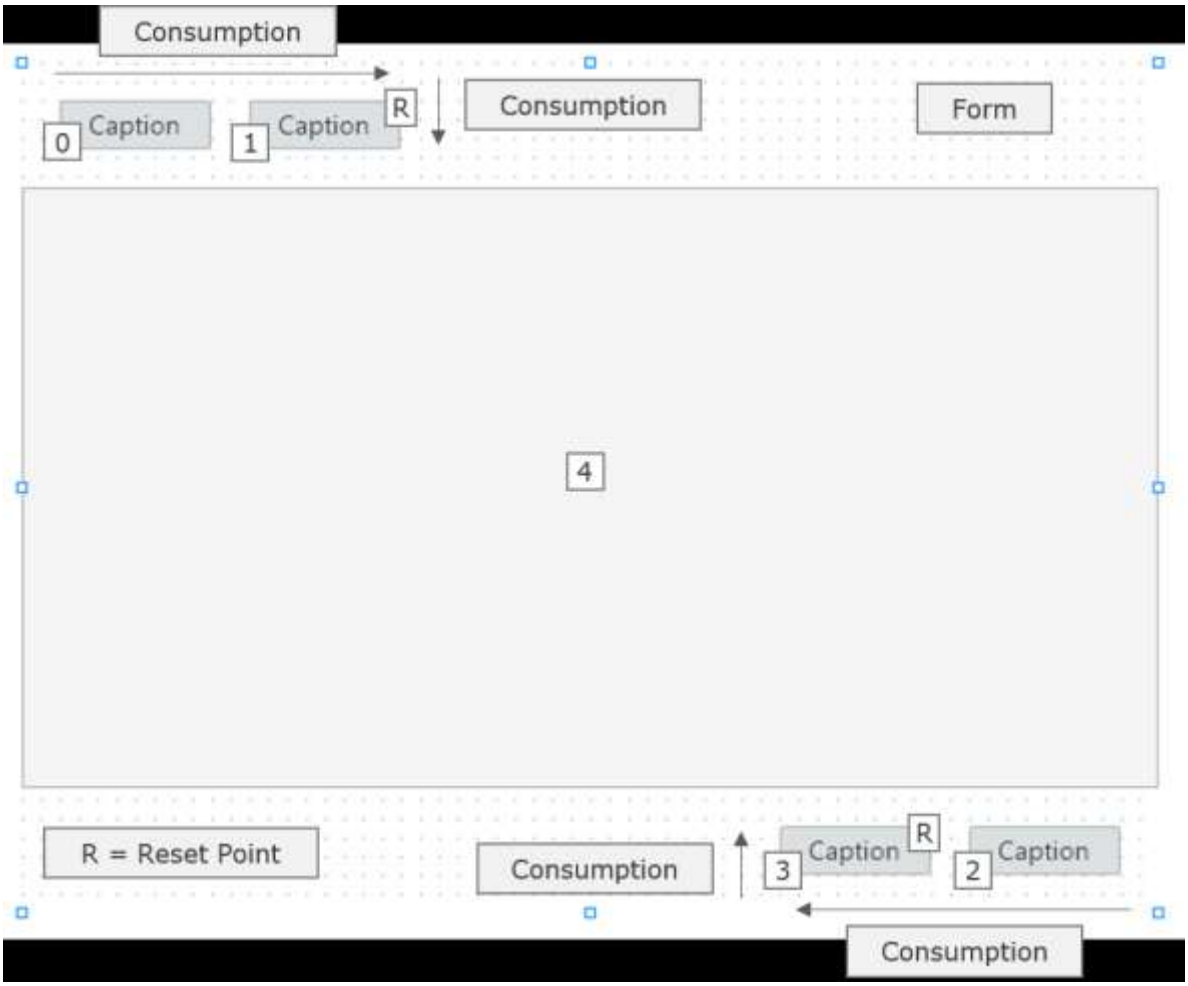
With these changes, the layout functionality reduced the height of the layout rectangle by the height of the second button (34 pixels) combined with the top margin of the second button (20 pixels), for a total reduction of 54 pixels.

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

Reset Points

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

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



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

Constraints and Stretching

The defined constraints for a control are **always** in effect, and any attempts to modify the dimensions of the control in a way that violates these constraints will result in the modification being adjusted so that it adheres to the applicable constraint. This makes constraints very useful when combined with the Layout.Stretch property options.

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

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

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

Layout Overflow and Responsive Layouts

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

Overflow	Consumption
loTop	lcTop
loLeft	lcLeft
loRight	lcRight
loBottom	lcBottom

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

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

This page intentionally left blank

Chapter 5
Using Server Requests

5.1 Server Request Architecture

Server Requests in Client Applications

Both visual and non-visual client applications are single-page web applications that are loaded in the web browser and stay loaded until the browser navigates to a different URL. Such an application is different from a traditional web site with a collection of individual web pages that are navigated using traditional URL links. Navigating to a different URL in aclient application will cause the application to be unloaded in the web browser, which is not the desired result for most situations.

Given this architecture, there needs to be a way for such an application to communicate with the web server in order to exchange data or content without causing an actual navigation or page load in the web browser. The name for this type of functionality in web browsers is called AJAX, which stands for "Asynchronous JavaScript and XML". The server request components wrap the AJAX functionality in the web browser for use with both visual and non-visual client applications.

All web server requests in client applications are asynchronous. This means that when a web server request is executed, the application will continue to execute and respond to user input while the request is being executed, and an event will be triggered when the request completes successfully or encounters an error.

> **Note**
> While AJAX was originally designed to be used primarily with XML data (per the naming), it can be used with any type of textual content or data, as well as with HTML form data (including file uploads).

Server Requests in Server Applications

Server applications also provide very similar server request components for executing web server requests to **other** web servers. This is useful for functionality like accessing payment processor APIs and other types of APIs in ways that aren't permissible or feasible for the client applications to execute directly from a web browser.

All web server requests in server applications are synchronous. This means that the server application will wait until the request is completely executed before executing any subsequent code after the server request execution. This makes it easier to deal with control flow and the debugging of server requests in server applications.

Web server requests in server applications have some additional capabilities that aren't present in client applications, such as the ability to specify the user agent (required for some APIs), the ability to specify a connection timeout, the ability to use binary streams for request and response content, and the ability to easily set up server requests that are sending multi-part content to a web server.

Core Components

The components that encapsulate the web server request functionality are:

TServerRequest
TServerRequest components can be dropped directly on a form or request handler at design-time in a visual client project or server project, or created at run-time in a visual/non-visual client project or server project. The TServerRequest component encapsulates a single web server request. The Method property specifies the HTTP

method (default rmGet) and the URL property specifies the URL for the request. Although the component will automatically populate all required request headers, you can specify additional request headers using the Headers property. The Execute method can be used to execute the request.

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

5.2 Executing a Server Request

The TServerRequest component is used to send/receive content to/from the web server. For client applications running in a web browser, any content sent using a server request must be textual and is normally sent to the web server as UTF-8-encoded text. For server applications, any content sent using a server request can be textual or binary, but textual data is also automatically encoded as UTF-8-encoded text. Both client applications and server applications can receive both textual and binary data, but client applications must deal with the received content as text and only server applications allow you to use streams to deal with binary data received in a response to a server request.

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

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

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

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

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

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

```
Name: Value
```

- Create and assign an event handler to the TServerRequest OnComplete event. This will ensure that you can determine when the request is complete.
- Call the TServerRequest Execute method to initiate the web server request.

> **Note**
> Remember that the Execute method is asynchronous when used in client applications running in a web browser, and synchronous when used in server applications running in the web server.

TServerRequest Example

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

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

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

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

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

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

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

TServerRequestQueue Example

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

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

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

```
   MyRequestQueue.Free;
end;

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

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

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

Cancelling a Server Request

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

HTTP Response Status Codes

There are defined constants that represent the common HTTP response status codes in the WebHTTP unit:

```
   HTTP_NONE = 0;

   HTTP_CONTINUE= 100;

   HTTP_OK = 200;
   HTTP_CREATED = 201;

   HTTP_MOVED_PERMANENTLY = 301;
   HTTP_FOUND = 302;
   HTTP_SEE_OTHER = 303;
   HTTP_NOT_MODIFIED = 304;
   HTTP_MOVED_TEMPORARILY = 307;

   HTTP_BAD_REQUEST = 400;
   HTTP_FORBIDDEN = 403;
   HTTP_NOT_FOUND = 404;
```

```
    HTTP_METHOD_NOT_ALLOWED = 405;
    HTTP_NO_LENGTH = 411;
    HTTP_PAYLOAD_TOO_LARGE = 413;
    HTTP_HEADERS_TOO_LARGE = 431;

    HTTP_INTERNAL_ERROR = 500;
    HTTP_NOT_IMPLEMENTED = 501;
    HTTP_SERVICE_UNAVAILABLE = 503;
    HTTP_VERSION_NOT_SUPPORTED = 505;
```

Chapter 6
Using Local Storage

6.1 Introduction

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

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

The standard component library surfaces the local storage via the TPersistentStorage component class, and automatically creates two instances of the TPersistentStorage class at application startup:

Storage Type	Instance Variable Name
Per-Session	SessionStorage
Local	LocalStorage

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

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

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

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

6.2 Saving Data To Local Storage

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

```
uses WebCore, WebComps;

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

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

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

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

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

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

6.3 Loading Data from Local Storage

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

```
uses WebCore, WebComps;

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

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

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

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

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

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

6.4 Detecting Local Storage Changes

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

```
uses WebCore, WebComps;

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

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

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

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

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

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

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

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

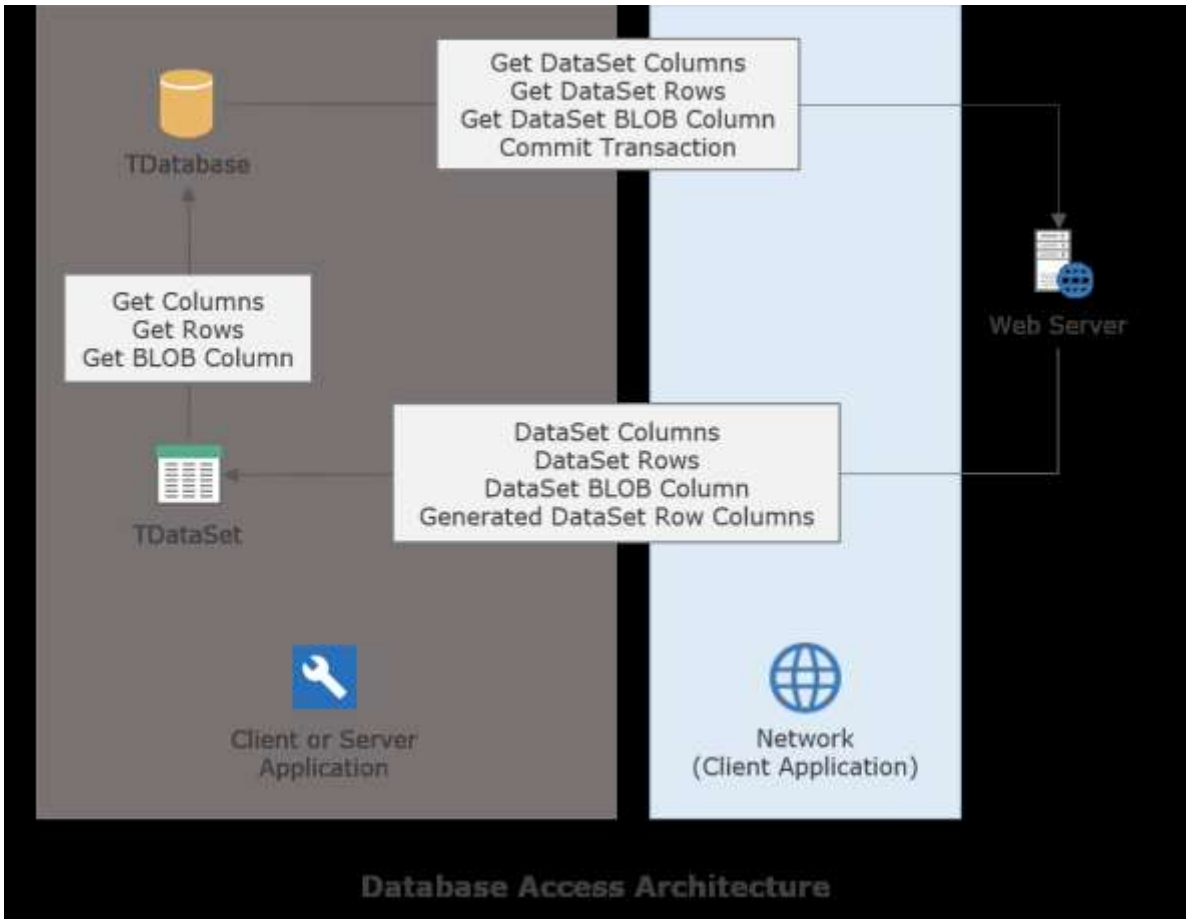
This page intentionally left blank

Chapter 7
Using Databases

7.1 Database Architecture

Elevate Web Builder includes database access functionality for easily loading data and then updating the same data using transactions.

The database access functionality has the following architecture:



The database access architecture uses a disconnected model where all data is cached locally, all database access on the web server is entirely stateless, and all database operations on the web server are performed optimistically as atomic transactions. All database access requests/responses use JSON as the underlying data format. Please see the Web Server Database Access and Web Server Database Access API topics for more information on how the database access works in the web server along with how the various database API requests and responses are structured.

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

Authentication

Database operations in client applications require an authenticated session in order to complete successfully.

Database operations in server applications are already operating under an authenticated session because server application execution also requires authentication. However, database access in server applications is unrestricted, and server applications do not use the effective privileges of the authenticated session in order to determine if any given database is accessible. Server applications behave as if they are executing as a "super user" with respect to database access. Please see the Web Server Authentication topic for more information on authentication in the web server.

Core Concepts

There are four core concepts in the database access functionality:

- **Loading DataSet Columns** - Normally the dataset columns are loaded/defined at design-time in the IDE, but it is possible to dynamically load the columns for a dataset at run-time. The returned JSON column information includes the column name, data type, length, and scale.

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

- **Loading DataSet BLOB Columns** - Any BLOB columns in dataset rows are dynamically loaded on-demand. For client applications, BLOB columns behave like string columns and contain a URL link that represents the web server request required to load the BLOB data. For server applications, BLOB columns are actually binary streams represented by a TStream instance that is accessible via the TDataColumn AsBlob property. The stream is loaded when it is first accessed and then cached as part of the in-memory row.

- **Transactions** - By default, transactions are automatically started and committed/rolled back as rows are inserted/saved, updated/saved, and deleted in any datasets contained within a database. Nested transactions are supported, so only the outermost commit operation actually results in a commit call to the web server. The automatic transaction handling can be turned off (see the TDatabase component below). The update of generated columns such as identity columns are supported in transactions, and any generated column values are echoed back to the client or server application's in-memory rows after a transaction successfully commits. Please see the Web Server Database Access topic for more information on how generated columns are handled in the web server.

Core Components

The database functionality contains several core components, all residing in the WebSession and WebData units in the standard component library.

TServerSession
A global TServerSession component instance called **Session** is auto-created at application startup for client applications. Server applications do not require authentication, and the WebSession unit and TServerSession component are not used with server applications.

In addition to this default singleton session instance, you can add explicit TServerSession instances to a visual client application project by dragging and dropping any server defined in the server manager on to a form or database designer surface. When the session is dropped on to the designer surface, a new TServerSession instance will be created and the relevant property information, such as the resource names for the server, will automatically be populated for the session. Please see the Using the Server Manager topic for more information.

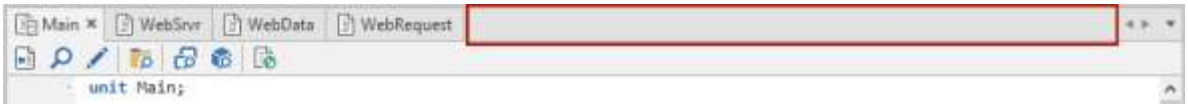
> **Note**
> The UserName property will be set to **Anonymous** when creating a TServerSession instance through the drag and drop method. The default Anonymous user in the web server does not have a password. However, if you change the UserName property to a different user name at design-time, be aware that the Password property is a public property only available at run-time and must be populated in code.

Each TDatabase component instance (see below) has a ServerSession property that refers to a TServerSession component instance and provides a way for the TDatabase instance to authenticate the client application/user before attempting to execute any database access requests. This singleton instance of the TServerSession component is automatically associated with any TDatabase component instances that do not contain an explicit TServerSession component instance reference. You can use the TDatabase ActiveServerSession property to determine the actual TServerSession instance being used by the database for authentication.

> **Note**
> If a session expires on the web server, an HTTP 403 error will be returned for any database access requests from the client application using the web server session. The TDatabase component will handle re-authentication and the estabishment of a new session, along with retrying the database access request, without any application intervention being required.

TDatabase
A global TDatabase component instance called **Database** is auto-created at application startup for both client and server projects. This singleton instance of the TDatabase component is used to keep track of all TDataSet (see below) instances that aren't associated with a specific TDatabase instance.

In addition to this default singleton database instance, you can add explicit TDatabase instances to a visual client or server application project by dragging and dropping any database defined in the server manager on to the tab gutter of the work area in the IDE:



When the database is dropped on the tab gutter of the work area, a new TDatabase (or descendant) instance will be created for the project, along with an associated source unit, and all of the defined datasets for the database will automatically be created as TDataSet instances in the new database instance. Please see the Using the Server Manager topic for more information.

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

TDataSet
TDataSet components can either be dropped directly on a form, request handler, or database at design-time in a visual client or server application project, or created at run-time in both visual and non-visual projects.

The Columns property contains the column definitions for the dataset. The column definitions for a dataset can be defined manually at design-time or load at run-time using the TDatabase LoadColumns method (via the TDatabase instance that contains the TDataSet instance) or the TDataSet LoadColumns method. The primary difference between the two lies with client applications: in client applications the TDatabase LoadColumns method transparently handles retrieving the column definitions from the web server, whereas the TDataSet LoadColumns method accepts a JSON string containing the column definitions, and leaves the details of where the JSON string originated up to the caller. With server applications, there exists an overloaded TDataSet LoadColumns method without the JSON string parameter, and both TDatabase and TDataSet LoadColumns methods perform the same function.

Rows must be loaded from the web server application at run-time using the TDatabase LoadRows method (via the TDatabase instance that contains the TDataSet instance) or the TDataSet LoadRows method. The primary difference between the two lies with client applications: in client applications the TDatabase LoadRows method transparently handles retrieving the rows from the web server, whereas the TDataSet LoadRows method accepts a JSON string containing the rows, and leaves the details of where the JSON string originated up to the caller. With server applications, there exists an overloaded TDataSet LoadRows method without the JSON string parameter, and both TDatabase and TDataSet LoadRows methods perform the same function.

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

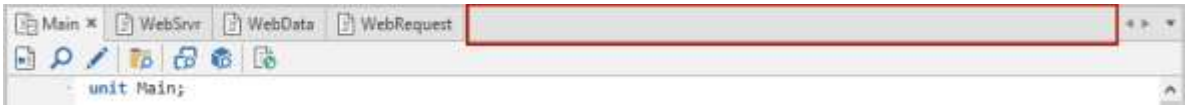
The TDataSet component also allows you to Insert, Update, and Delete rows, as well as Find and Sort rows.

7.2 Creating and Using Databases

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

Creating a Database at Design-Time

The easiest way to create a database is by using the server manager in the IDE to define a database and its contained datasets. Once a database has been defined in the server manager, you can easily add the database to an existing client or server application project by dragging and dropping the database on to the tab gutter of the work area in the IDE:



When the database is dropped on the tab gutter of the work area, a new TDatabase (or descendant) instance will be created for the project, along with an associated source unit, and all of the defined datasets for the database will automatically be created as TDataSet instances in the new database instance. Please see the Using the Server Manager topic for more information.

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

Database Authentication

Each TDatabase component instance in a client application has a ServerSession property that refers to a TServerSession component instance and provides a way for the TDatabase instance to authenticate the client application/user before attempting to execute any database access requests. This singleton instance of the TServerSession component is automatically associated with any TDatabase component instances that do not contain an explicit TServerSession component instance reference. You can use the TDatabase ActiveServerSession property to determine the actual TServerSession instance being used by the database for authentication.

> **Note**
> Server applications do not require authentication, and the WebSession unit and TServerSession component are not used with server applications.

Database Request Queue

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

If a session expires on the web server, an HTTP 403 error will be returned for any database access requests from the client application using the web server session. The TDatabase component will handle re-authentication and the estabishment of a new session, along with retrying the database access request, without any application intervention being required. Authentication is performed in client applications using an out-of-band server request that is separate from the request queue used for normal database access requests.

> **Note**
> Server applications do not require server requests for database access and a request queue is not used with TDatabase instances in server applications.

Transactions

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

Database Parameters

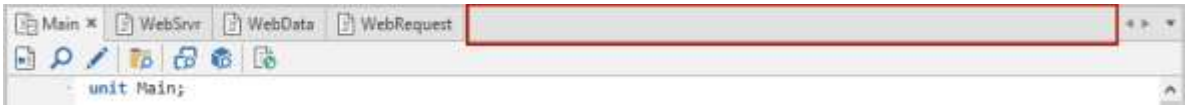
You can use the TDatabase Params property to specify database-specific parameters that will be passed as URL parameters with all database requests originating from the database. This is useful for situations where you want to tag all database requests with application-specific information. The Params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

7.3 Creating and Loading DataSets

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

Creating a DataSet at Design-Time

The easiest way to create a dataset is by using the server manager in the IDE to define a database and its contained datasets/commands. Once a dataset has been defined under a database in the server manager, you can easily add the dataset to an existing application by simply dragging it from the server manager and dropping it on a form, request handler, or database designer surface. The relevant property information, including the column definitions, will automatically be populated for the dataset. A database defined in the server manager can be used to create a database in an existing client or server application project by dragging and dropping the database on to the tab gutter of the work area in the IDE:



When the database is dropped on the tab gutter of the work area, a new TDatabase (or descendant) instance will be created for the project, along with an associated source unit, and all of the defined datasets for the database will automatically be created as TDataSet instances in the new database instance. Please see the Using the Server Manager topic for more information.

If you do not wish to use the server manager to create a dataset, you can also create a new dataset by dragging a TDataSet component from the component library and dropping it on a form, request handler, or database designer surface. Please see the Using the Designer topic for more information on adding a component to a designer instance. Once you have dropped the TDataSet component on a form, request handler, or database designer surface, you can manually define the columns in the dataset by double-clicking on the TDataSet's Columns property. This will launch the Columns Editor directly under the component inspector, and you can then use the Columns Editor to add, edit, or delete the columns in the dataset. Please see the Using the Component Inspector topic for more information on how to modify properties in the IDE.

Creating a DataSet at Run-Time

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

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

```
      Save;
      Insert;
      Columns['Abbrev'].AsString:='NY';
      Save;
      end;
end;
```

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

Loading a DataSet at Run-Time

As seen in the above example, you can insert rows directly into a dataset at run-time without interacting with the web server. However, most applications will need to load the rows into a dataset using the web server. For both client and server applications, there are two different ways to load rows into a dataset at run-time: the TDatabase LoadRows method and the TDataSet LoadRows method.

TDatabase LoadRows Method

The TDatabase LoadRows method should be used when you want to load the dataset rows from the web server. For client applications, this method automatically handles the server request to the web server. For server applications, the rows are loaded directly from the web server using native API calls.

The TDatabase component uses the following properties to load the dataset rows from the web server:

- TDatabase BaseURL
  This public property is a read-only, calculated property that returns the concatenation of the TServerSession BaseURL and DatabasesResource properties of the server session referenced in the ActiveServerSession property with the TDatabase DatabaseName property:

  ```
  <TServerSession.BaseURL>/<TServerSession.DatabasesResource>/<TDatabase.Databa
        seName>
  ```

  This property is used by a TDatabase instance in client applications to build the URL used for the server request to the web server. This property is not used in server applications, and is unavailable.

- TDatabase DatabaseName
  This property defaults to the same value as the TDatabase component's Name property, but is automatically populated for you if you use the drag-and-drop method of creating a TDatabase at design-time. This property is used to identify the database in database access requests and must match the name of an existing database defined on the web server.

- TDatabase Params
  This property is a string list (TStrings) of "name=value" pairs that represents any application-specific parameters to be used with all dataset commands executed for the database. Please see the Web Server Database Access topic for more information on how dataset commands are defined and how they use input parameters.

- TDataSet DataSetName
  This property defaults to the same value as the TDataSet component's Name property, but is automatically populated for you if you use the drag-and-drop method of creating a TDataSet at design-time. This property is used to identify the dataset in database access requests and must match the name of an existing dataset defined on the web server within the database specified by the TDatabase DatabaseName property (see above).

- TDataSet Params
  This property is a string list (TStrings) of "name=value" pairs that represents the parameters used along with the parameters defined in the TDatabase Params property (see above) by the web server to populate any input parameters in the **Select** dataset command. Please see the Web Server Database Access topic for more information on how dataset commands are defined and how they use input parameters.

As an example, consider a database and dataset that are defined as follows on the web server:

```
Database Name: Production

DataSet Name: CustomerOrders

Select DataSet Command:

SELECT * FROM custord
WHERE CustomerID=:CustomerID
```

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

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

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

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

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

      CustomerOrders.Params.Add('CustomerID='+Customer.Columns['CustomerID'].AsStri
      ng);
   LoadRows(CustomerOrders);
end;
```

After the rows are successfully retrieved from the web server, the TDatabase LoadRows method will automatically open the dataset using the TDataSet Open method and then automatically call the TDataSet LoadRows method to load the rows into the in-memory cache of the dataset.

TDataSet LoadRows Method

For both client and server applications, the TDataSet LoadRows method directly accepts the dataset rows as a JSON

string. This means that this method is more useful for situations where the dataset rows are stored in memory or local storage as a JSON string and need to be directly loaded from one of those locations. It is recommended that you always use the TDatabase LoadRows method for loading rows from a web server and not resort to making custom server requests.

> **Note**
> The LoadRows method requires that the dataset be open prior to being called. Use the Open method to open the dataset.

Tracking Load Operations

The TDataSet BeforeLoad event is fired before the dataset load actually begins. To prevent the load from occurring, return False as the result in an event handler for this event.

If a dataset load server request was sent to the web server and was not successful due to the web server returning an HTTP status code other than 200 (OK), the OnLoadError event will be fired and will include the error message. If an event handler is not defined for the OnLoadError event, then an exception will be raised with the error message. If a load fails for any reason, then the load request is placed in a pending requests queue for the database. This queue ensures that the database requests can be retried and, when retried, are sent to the web server in the order in which they occurred. You can see if there are any pending database requests by examining the TDatabase NumPendingRequests property. If the NumPendingRequests property is greater than 0, then there are database access requests that need to be retried at some point. Use the TDatabase RetryPendingRequests method to retry any pending database access requests, and the TDatabase CancelPendingRequests method to cancel any pending database access requests.

The TDataSet AfterLoad event is fired after the dataset load completes successfully. If there were any errors during the load process, then this event handler will not get called.

7.4 Navigating DataSets

The TDataSet component provides several methods for navigating the rows present in the underlying dataset, as well as properties for obtaining information about the current row position and reading data from the current row.

Moving the Row Pointer

To move the row pointer to a different position in the dataset, use the TDataSet First, Prior, Next, Last, MoveTo, and MoveBy methods. Use the TDataSet BOF, EOF, and RowNo properties to obtain information about the current row position.

The following example navigates from the beginning of a dataset to the end, appending each order ID to a string:

```
var
   OrderIDs: String='';
begin
   with CustomerOrders do
      begin
      First;
      while (not EOF) do
         begin
         if (OrderIDs='') then
            OrderIDs:=Columns['OrderID'].AsString
         else
            OrderIDs:=OrderIDs+', '+Columns['OrderID'].AsString;
         Next;
         end;
      end;
end;
```

Bookmark Operations

Sometimes it is necessary to save the current row pointer, perform some operations that may or may not move the row pointer, and then return to the saved row pointer. The TDataSet SaveBookmark, GotoBookmark, and FreeBookmark methods provide the bookmark functionality for datasets. Bookmarks include a non-volatile row ID and BOF/EOF information so that a row pointer can be restored even when the active sort has been changed. The only case when a row pointer cannot be restored is when the row represented by the bookmark has been deleted.

Note
Bookmarks are automatically pushed and popped from an internal bookmark stack for the dataset, so nested calls to SaveBookmark and GotoBookmark/FreeBookmark will automatically work properly as long as the number of GotoBookmark/FreeBookmark calls matches the number of SaveBookmark calls. Also, GotoBookmark and FreeBookmark are mutually-exclusive: both methods free the active bookmark, but only the GotoBookmark method actually tries to navigate to the active bookmark before freeing it.

The following example saves the current row pointer as a bookmark, updates a column in all of the rows, and then restores the row pointer by calling GotoBookmark:

```
procedure TOrderEntryDlg.UpdateLineNumbers;
begin
   with CustomerItems do
      begin
```

```
      DisableControls;
      try
         SaveBookmark;
         try
            First;
            while (not EOF) do
               begin
               Update;
               Columns['LineNo'].AsInteger:=RowNo;
               Save;
               Next;
               end;
         finally
            GotoBookmark;
         end;
      finally
         EnableControls;
      end;
      end;
end;
```

Reading Column Values

The TDataSet Columns property allows you to read the column values for the current row. You can access a column in the Columns property by its index or by its name via the TDataColumns Column property. However, since the Column property is the default property for the TDataColumns object, you can omit it when referencing the Columns property. The following example loops through all columns in a dataset and appends their name to a string:

```
var
   I: Integer;
   ColumnNames: String='';
begin
   with CustomerOrders do
      begin
      for I:=0 to Columns.Count-1 do
         begin
         if (ColumnNames='') then
            ColumnNames:=Columns[I].Name
         else
            ColumnNames:=ColumnNames+','+Columns[I].Name
         end;
      end;
end;
```

Each TDataColumn object present in the TDataSet Columns property has several As* properties that allow you to access the data in the column for the current row as a particular type. Type conversions are performed automatically wherever necessary. However, certain type conversions are impossible and will, if attempted, cause an exception to be raised. For example, the following code will cause an exception to be raised because the OrderDate column, which has a type of dtDate, cannot be converted to a Boolean value:

```
begin
   with CustomerOrders do
      Result:=Columns['OrderDate'].AsBoolean;
end;
```

To determine if a column is Null, you can use the TDataColumn Null property.

Tracking Navigation Operations

The TDataSet BeforeScroll event is fired before the dataset's row pointer moves during navigation. To prevent the navigation from occurring, return False as the result in an event handler for this event.

The TDataSet AfterScroll event is fired after the dataset's row pointer is moved.

The following TDataSet property assignments cause the BeforeScroll and AfterScroll events to be triggered:

RowID
RowNo

The following TDataSet methods cause the BeforeScroll and AfterScroll events to be triggered:

First
Prior
Next
Last
MoveBy
MoveTo
Find
Sort
GotoBookmark

7.5 Searching and Sorting DataSets

The TDataSet component provides several methods for searching and sorting the rows present in the underlying dataset, as well as properties for obtaining information about the active sort.

Sorting the Rows

To sort the rows in a dataset, use the Sort method. To specify the columns to sort, assign the desired value to the TDataColumn SortDirection property in the order that reflects the column order of the desired sort. Use the TDataSet SortCaseInsensitive property to specify that the sort should be case-insensitive, and the SortLocaleInsensitive property to specify that the sort should be locale-insensitive. The default value for both properties is False.

The following example sorts the Products dataset based upon descending list price:

```
begin
   with Products do
      begin
      Columns['ListPrice'].SortDirection:=sdDescending;
      Sort;
      end;
end;
```

Once a sort has been established, the TDataSet Sorted will return True and the dataset will automatically keep the rows sorted accordingly as rows are inserted, updated, or deleted. The TDataColumn SortIndex property can be examined to determine where a column resides in the active sort. To clear an existing sort, simply assign a value of sdNone to the SortDirection property of all sorted columns.

Searching for a Row

The TDataSet Find method allows you to search the rows in the dataset for a particular set of column values. If there is a sort active on the dataset, then it will be used for satisfying the Find operation if the ColumnsToSearch, CaseInsensitive, and LocaleInsensitive parameters for the Find method match the active sort. For example, the following example sorts the Products dataset by the ProductID column and then executes a case-insensitive Find operation on the ProductID column for the 'PEN-BP-12PK' product ID:

```
begin
   with Products do
      begin
      Columns['ProductID'].SortDirection:=sdAscending;
      SortCaseInsensitive:=True;
      Sort;
      if Find(['ProductID'],['PEN-BP-12PK'],False,True) then
         Result:=True
      else
         Result:=False;
      end;
end;
```

Pass True as the third NearestSearch parameter to the Find method in order to perform a search for the row with the column values that are nearest to the specified Find values.

7.6 Updating DataSets

The TDataSet component provides several methods for inserting, updating, and deleting rows in the underlying dataset, as well as properties for reading both the current and old column values from the current row.

Inserting New Rows

Inserting a new row in a dataset is a three-step process. First, use the TDataSet Insert method to put the dataset into the insert state. This will:

- Fire the BeforeInsert event handler, if one is defined. To prevent the insert from occurring, return False as the result in the event handler.
- If the OwnerDatabase's AutoTransactions property is True (the default), then start a new transaction and then create a new row. If the Append flag (default False) is passed to the Insert method, then the new row will be appended to the end of the dataset. Otherwise, the new row will be inserted at the current row pointer in the dataset.
- Fire the OnInitrow event handler, if one is defined. The OnInitRow event handler allows the application to assign values to the columns in the new row without causing any of the columns, or the row, to be flagged as modified. This is a good place to assign default values for columns.
- Change the TDataSet State property to dsInsert once the dataset is in the insert state.
- Fire the AfterInsert event handler, if one is defined.

Once the dataset is in the insert state, you can use the Columns property to read or assign new values to the various columns in the row using the TDataColumn As* properties. When a column is assigned a new value, its Modified property is set to True.

When all column modifications have been made, use the Save method to complete the insert. This will:

- Fire the BeforeSave event handler, if one is defined. To prevent the save from occurring, return False as the result in the event handler.
- Save the row in the dataset, logging the insert if a transaction is in progress. At this point, the Modified property for the columns in the row will be reset to False.
- Fire the AfterSave event handler, if one is defined.
- Update any active sort and change the TDataSet State property to dsBrowse to reflect that the dataset is now in the browse state.

> **Note**
> With no active sort, rows are always sorted by their actual insertion order, so even if the Insert method was called without the Append flag, the newly-inserted row will move to the end of the dataset after the Save method completes.

- If the OwnerDatabase's AutoTransactions property is True (the default), then commit the active transaction.

If you want to cancel the insert operation, you can use the TDataSet Cancel method. This will:

- Fire the BeforeCancel event handler, if one is defined. To prevent the cancel from occurring, return False as the result in the event handler.
- Discard the row. The row pointer will return to the row pointer that was active prior to the Insert method being called.
- Fire the AfterCancel event handler, if one is defined.
- Change the TDataSet State property to dsBrowse to reflect that the dataset is now in the browse state.
- If the OwnerDatabase's AutoTransactions property is True (the default), then roll back the active transaction.

The following example inserts a new product into the Products dataset:

```
begin
   with Products do
      begin
      Products.Insert; // Required to avoid conflict with Insert system
      function
      Columns['ProductID'].AsString:='PHONE-HEADSET';
      Columns['Description'].AsString:='Hands-free phone handset';
      Columns['ListPrice'].AsFloat:=15.00;
      Columns['Shipping'].AsFloat:=2.00;
      Save;
      end;
end;
```

Updating Existing Rows

Updating an existing row in a dataset is a three-step process. First, use the TDataSet Update method to put the dataset into the update state. This will:

- Fire the BeforeUpdate event handler, if one is defined. To prevent the update from occurring, return False as the result in the event handler.
- If the OwnerDatabase's AutoTransactions property is True (the default), then start a new transaction.
- Change the TDataSet State property to dsUpdate once the dataset is in the update state.
- Fire the AfterUpdate event handler, if one is defined.

Once the dataset is in the update state, you can use the Columns property to read or assign new values to the various columns in the row using the TDataColumn As* properties. When a column is assigned a new value, its Modified property is set to True. You can use the TDataColumn OldValue property to access the value of any column before any new assignments were made to the row.

When all column modifications have been made, use the Save method to complete the update. This will:

- Fire the BeforeSave event handler, if one is defined. To prevent the save from occurring, return False as the result in the event handler.
- Save the row in the dataset, logging the update if a transaction is in progress. At this point, the Modified property for the columns in the row will be reset to False.
- Fire the AfterSave event handler, if one is defined.

- Update any active sort and change the TDataSet State property to dsBrowse to reflect that the dataset is now in the browse state.

> **Note**
> If any of the columns modified during the update are part of the active sort, then the row will automatically move to the correct position in the active sort.

If you want to cancel the update operation, you can use the TDataSet Cancel method. This will:

- Fire the BeforeCancel event handler, if one is defined. To prevent the cancel from occurring, return False as the result in the event handler.
- Discard any modifications to the row. The row pointer will stay in the same location.
- Fire the AfterCancel event handler, if one is defined.
- Change the TDataSet State property to dsBrowse to reflect that the dataset is now in the browse state.
- If the OwnerDatabase's AutoTransactions property is True (the default), then roll back the active transaction.

The following example finds a product in the Products dataset and updates its shipping cost:

```
begin
   with Products do
      begin
      if Find(['ProductID'],['PHONE-HEADSET'],False,True) then
         begin
         Update;
         Columns['Shipping'].AsFloat:=1.80;
         Save;
         end;
      end;
end;
```

Deleting Existing Rows

Deleting an existing row in a dataset can be done by calling the TDataSet Delete method. This will:

- Fire the BeforeDelete event handler, if one is defined. To prevent the delete from occurring, return False as the result in the event handler.
- If the OwnerDatabase's AutoTransactions property is True (the default), then start a new transaction and delete the existing row.
- Fire the AfterDelete event handler, if one is defined.

The following example finds a product in the Products dataset and deletes it:

```
begin
   with Products do
      begin
      if Find(['ProductID'],['FLASH-USB-16GB'],False,True) then
         Delete;
      end;
end;
```

7.7 Transactions

While datasets can be updated without using transactions, doing so causes all updates to be merged into the in-memory rows in the dataset in the client or server application and unable to be committed to a physical database accessible to the web server. Using transactions allows all updates to a dataset to be logged so that the updates can then be committed to the web server and reflected in a physical database.

All transaction properties, methods, and events are contained with the TDatabase component in the WebData unit. The AutoTransactions property is True, by default, and controls whether transactions are automatically started and committed/rolled back as the datasets are updated. Please see the Updating DataSets topic to see how the automatic transactions interact with the various dataset insert, update, and delete operations. The methods for starting, committing, and rolling back transactions are the StartTransaction, Commit, and Rollback methods. Transactions can be nested, so the TDatabase InTransaction and TransactionLevel properties reflect whether a transaction is active and at what level, respectively. If the TransactionLevel property is -1, then no transactions are active.

Starting a Transaction

Use the TDatabase StartTransaction method to start a transaction. This will increment the current transaction level. All row inserts, updates, and deletes taking place in any owned datasets will be automatically logged as part of the current transaction.

Committing a Transaction

Use the Commit method to commit the current transaction. This will:

- Fire the BeforeCommit event handler, if one is defined. To prevent the commit from occurring, return False as the result in the event handler.
- If the current transaction level, as reflected by the TDatabase TransactionLevel property, is 0, commit the transaction on the web server.
- If the current transaction level is greater than 0, then append all operations in the current transaction to the next lower transaction.
- Decrement the current transaction level. If the transaction level is -1, then the TDatabase InTransaction property is set to False.
- If the transaction level is greater than -1, then immediately fire the AfterCommit event handler, if one is defined.
- If the transaction level is -1, wait for a result from the transaction commit operation on the web server. If the transaction commit was not successful due to the web server returning an HTTP status code other than 200 (OK), the OnCommitError event will be fired and will include the error message. If an event handler is not defined for the OnCommitError event, then an exception will be raised with the error message. If a commit fails for any reason, then the transaction being committed is placed in a pending requests queue. This is also true for general database access requests such as dataset load requests. This queue ensures that the database requests can be retried and, when retried, are sent to the web server in the order in which they occurred. You can see if there are any pending database requests by examining the TDatabase NumPendingRequests property. If the NumPendingRequests property is greater than 0, then there are commit and/or dataset load requests that need to be retried at some point. Use the TDatabase RetryPendingRequests method to retry any pending database requests, and the TDatabase CancelPendingRequests method to cancel any pending database requests. If the transaction commit was successful, then fire the AfterCommit event handler, if one is defined.

Transaction Commit Properties

The TDatabase component uses the following properties when committing a transaction:

- TDatabase BaseURL
  This public property is a read-only, calculated property that returns the concatenation of the TServerSession BaseURL and DatabasesResource properties of the server session referenced in the ActiveServerSession property with the TDatabase DatabaseName property:

  ```
  <TServerSession.BaseURL>/<TServerSession.DatabasesResource>/<TDatabase.Databa
        seName>
  ```

  This property is used by a TDatabase instance in client applications to build the URL used for the server request to the web server. This property is not used in server applications, and is unavailable.

- TDatabase DatabaseName
  This property defaults to the same value as the TDatabase component's Name property, but is automatically populated for you if you use the drag-and-drop method of creating a TDatabase at design-time. This property is used to identify the database in database access requests and must match the name of an existing database defined on the web server.

- TDatabase Params
  This property is a string list (TStrings) of "name=value" pairs that represents any application-specific parameters to be used with all dataset commands executed for the database. Please see the Web Server Database Access topic for more information on how dataset commands are defined and how they use input parameters.

Rolling Back a Transaction

Use the Rollback method to roll back the current transaction. This will:

- Fire the BeforeRollback event handler, if one is defined. To prevent the rollback from occurring, return False as the result in the event handler.

- Undo all operations that have taken place in the current transaction. If there are any exceptions during this process, the OnRollbackError event will be fired and will include the error message. If an event handler is not defined for the OnRollbackError event, then an exception will be raised with the error message.

  Note
  It is highly unlikely that an exception will ever be raised when a transaction is being rolled back, but it is possible that a catastrophic error could cause such an exception.

- Decrement the current transaction level. If the transaction level is -1, then the TDatabase InTransaction property is set to False.

- Fire the AfterRollback event handler, if one is defined.

The following example starts a transaction, deletes all rows, and then commits the transaction.:

```
begin
   Database.StartTransaction;
   with Products do
      begin
      while (RowCount > 0) do
         Delete;
```

```
      end;
   Database.Commit;
end;
```

Note
If you attempt to call the TDatabase Commit or Rollback methods when there are no active transactions (InTransaction property is False), then an exception will be raised.

7.8 Responding to DataSet Changes

It is important that one be able to respond to various changes to dataset columns and rows, both for updating control states and for implementing concepts such as master-detail linkages. The TDataSet OnStateChange event is used to track when the TDataSet State changes. The TDataSet OnRowChanged event is used to track when the active row in the dataset changes, or when a column in the active row changes.

State Changes

Define an event handler for the TDataSet OnStateChange event in order to track when the dataset state changes. The following list details the TDataSet methods that cause the dataset state to change, along with the state after the method completes:

Method	After State
Open	dsBrowse
Close	dsClosed
CheckBrowseMode	dsBrowse (if result is True)
Find	dsBrowse
Insert	dsInsert
Update	dsUpdate
Save	dsBrowse
Cancel	dsBrowse

The following example shows an OnStateChange event handler that displays the state of a dataset called "Vendors" in a label on the current form:

```
procedure TMyForm.VendorsStateChange(Sender: TObject);
begin
   case Vendors.State of
      dsClosed:
         VendorStateLabel.Caption:='Closed';
      dsBrowse:
         VendorStateLabel.Caption:='Browse';
      dsInsert:
         VendorStateLabel.Caption:='Insert';
      dsUpdate:
         VendorStateLabel.Caption:='Update';
      end;
end;
```

Row Changes

Define an event handler for the TDataSet OnRowChanged event in order to track when the active row in the dataset changes. The OnRowChanged event is fired when any column in the active row is changed due to a modification, or when the active row changes due to the row pointer moving. If the OnRowChanged event was fired in response to a column modification, then the Column parameter in the event handler will contain an instance of the TDataColumn that was modified. If the OnRowChanged event was fired in response to the entire active row changing, then the Column parameter will be nil.

The following TDataColumn properties and methods will cause the OnRowChanged event to fire with a non-nil Column parameter:

AsString
AsBoolean
AsInteger
AsFloat
AsDate
AsTime
AsDateTime
Clear

The following TDataSet properties and methods will cause the OnRowChanged event to fire with a nil Column parameter:

RowID
LoadRows
Sort
EnableControls
First
Prior
Next
Last
MoveBy
MoveTo
GotoBookmark
Find
Insert
Save
Cancel
Delete

The following TDatabase properties and methods will cause the OnRowChanged event to fire with a nil Column parameter:

LoadRows
Rollback

Responding to row changes is important for updating related controls in the user interface. The following example shows an OnRowChanged event handler that responds to row changes by calling methods that update both buttons and labels:

```
procedure TMasterDetailForm.CustomerOrdersRowChanged(Sender: TObject;
                                                     Column: TDataColumn);
begin
   if (Column=nil) then
      begin
      UpdateOrderButtons;
      UpdateOrderLabels;
      end;
end;
```

Responding to row changes is also important for concepts such as master-detail links. The following example shows an OnRowChanged event handler that responds to row changes for loading a detail dataset as the master row changes:

```
procedure TMasterDetailForm.LoadOrders;
```

```
begin
   CustomerOrders.Params.Clear;

      CustomerOrders.Params.Add('CustomerID='+Customer.Columns['CustomerID'].AsStri
      ng);
   Database.LoadRows(CustomerOrders);
end;

procedure TMasterDetailForm.CustomerRowChanged(Sender: TObject;
                                               Column: TDataColumn);
begin
   if (Column=nil) then
      begin
      UpdateCustomerButtons;
      LoadOrders;
      end;
end;
```

7.9 Binding Controls to DataSets

The controls in the standard component library can be used in both an unbound and bound manner. A control is considered bound when it is explicitly attached to a dataset and one or more columns in the dataset. Most controls bind to a specific dataset column, while certain controls like the TGrid control can bind to multiple dataset columns.

Once a control is bound to a dataset, it will automatically update its contents in response to changes in the dataset. For example, if you insert a new row in the dataset using the TDataSet Insert method, then the control will automatically repopulate with the value from the new row.

Binding to a DataSet

To bind a control to a specific dataset, assign an existing TDataSet instance to the DataSet property of the control. This can be done at design-time or run-time.

> **Note**
> If the TDataSet instance that is assigned to the DataSet property is deleted, the control will automatically assign a value of nil to the DataSet property and the control will become unbound.

However, simply assigning the DataSet property is insufficient for binding a control to a dataset - you must also specify which column in the dataset to bind to. For most controls, this is done by assigning a column name to the DataColumn property. For the TGrid component, you must assign a column name to the DataColumn property of each TGridColumn in the grid that you wish to be bound to the dataset.

> **Note**
> The TGrid component allows you to to mix bound and un-bound columns within the same grid control.

Auto-Editing of Bound Controls

The value of the TDataSet AutoEdit property determines whether modifications to the contents of bound controls are allowed when the dataset is not in an editable state (TDataSet State property is dsInsert or dsUpdate). If the AutoEdit property is True, then any modification to a bound control will cause the attached dataset to insert a new row if the dataset is empty, or begin updating the current row if the dataset is not empty. If the AutoEdit property is False, then bound controls are effectively read-only until either a new row is inserted or an existing row is updated in the dataset.

> **Note**
> The TGrid control has two properties that can still enable a user to insert or delete rows in a bound grid. They are the AllowInserts and AllowDeletes properties, respectively. Be sure to set these properties to False if you do not want to allow a user to automatically insert or delete rows by using keystrokes in the grid.

Auto-Editing and Read-Only Columns

Dataset columns can be defined as calculated or read-only using the TDataColumn Calculated and ReadOnly properties, respectively. Any controls bound to a calculated or read-only column will not be editable, and will behave as though the dataset's AutoEdit property is set to False.

7.10 Calculated Columns

As discussed in the Creating and Loading DataSets topic, dataset columns are normally defined automatically when dragging a dataset from the server manager in the IDE and dropping the dataset on a form, request handler, or database designer surface, or they can be loaded at runtime from the web server via the TDatabase LoadColumns method.

However, in some cases you may want to define columns that derive their contents from a calculation. These types of columns are, of course, called calculated columns. Creating a calculated column is very simple:

- Create the column as you normally would, using the Add method of the TDataSet Columns.
- Set the new column's Calculated property to True.
- Define an event handler for the TDataSet OnCalculateRow event that executes the calculation code and assigns a value to the new calculated column.

Whenever a column in a row is updated, the OnCalculateRow event handler will be triggered so that any calculated columns can be re-computed for that row.

Any editable controls bound to a calculated column will automatically be read-only.

The following is an example of creating a calculated column that shows concatenated information from two other columns in the dataset:

```
procedure TForm1.Form1Create(Sender: TObject);
begin
   with Albums.Columns.Add do
      begin
      Name:='ArtistYear';
      DataType:=dtString;
      Length:=60;
      Calculated:=True;
      end;
   Albums.OnCalculateRow:=AlbumsCalculateRow;
end;

procedure TForm1.AlbumsCalculateRow(Sender: TObject; Column: TDataColumn);
begin
   Albums.Columns['ArtistYear'].AsString:=Albums.Columns['Artist'].AsString+
            ' ('+Albums.Columns['Year'].AsString+')';
end;
```

Note
Do not attempt to programmatically modify a calculated column outside of an OnCalculateRow event handler. Attempting to do so will result in an error. Also, you **cannot** modify non-calculated columns in an OnCalculateRow event handler.

Chapter 8
Language Reference

8.1 Introduction

Elevate Web Builder uses an Object Pascal dialect for its core language that is very close to the Object Pascal language used by Embarcadero RAD Studio and Delphi. Object Pascal was chosen as the language because it is a very easy language to learn due to its very English-like keywords, and because it is structured and strongly-typed, allowing the applications to avoid run-time errors that can cause problems for un-typed languages like JavaScript (the target code of the compiler for client applications).

The following are the rules governing the basic structure of the language.

Character Set

The Unicode character set is used for all language elements. Please see the Literals and Identifiers sections below for information on the restrictons to the allowed characters for both.

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

Case-Sensitivity

The language is not case-sensitive. Identifiers and other language keywords are always compared without considering case.

Statement Terminator

The semicolon (;) is the code statement terminator character in the language. It is used to indicate the ending of a statement, even if the statement spans more than one physical line:

```
begin
   if True then
      ShowMessage('It''s true !!!')
   else
      ShowMessage('It''s false !!!');
end;
```

In the above case, the extra line breaks are for formatting purposes only. However, you should always strive to format your code according to established formatting rules for the Object Pascal language, and such line breaks are very important for readability of your code.

Comments

The language supports both single-line comments using two slashes (//) or multi-line comments using left and right braces ({}):

```
begin
   // This piece of code needs some work
   if (not True) then
      BlowUpTheApplication
   else
      begin
      { Whew, we avoided blowing up the application,
        so let's continue on a more reasonable path }
      HandleTheSpecialCase;
      end;
end;
```

Literals

Literal values are specified as follows:

Value Type	Example
Numbers	100 1200.42 -39.00
Boolean	True false
Strings	'This is a string literal' 'This is a '+' concatentated '+' string literal'
Characters	'a' #27
Arrays	['This','is','a','string','array','literal'] [100,2,45]
References	nil

Identifiers

An identifier is the name of any system-declared or user-declared object in an application, such as units, constants, types, variables, or procedures/functions. Identifiers may begin with an underscore (_) or a letter (a-z, A-Z), and may contain an underscore, a letter, or a digit (0-9).

Reserved Words

The following are the list of reserved words. These words should not be used as identifiers:

```
abstract
and
array
as
async
begin
break
case
class
const
constructor
```

```
contains
continue
default
destructor
div
do
downto
else
end
except
exit
external
finalization
finally
for
function
if
implementation
inherited
initialization
interface
is
mod
not
object
of
on
or
out
override
private
procedure
program
property
protected
public
raise
read
record
repeat
shl
shr
then
to
try
type
unit
until
uses
var
virtual
while
with
write
xor
```

Syntax Diagrams

In the language reference syntax diagrams, angle brackets (<>) represent a language element and brackets ([]) represent an optional language element.

8.2 Defines

Elevate Web Builder supports basic compiler define functionality. Compiler defines are symbols used to conditionally include or exclude code in the compilation process, and can be tested at compile-time to make such a determination. Compiler defines are taken into account during the parsing phase of the compilation process.

Defining Symbols

You can create a compiler define using the following syntax:

```
{$DEFINE <Symbol>}
```

Once a symbol has been defined, it will be effective for the remaining code in the current unit. Defining a symbol that is already defined does nothing.

Warning
Compiler defines are **not** nested. If a symbol is re-defined (it was already defined), and then un-defined, the result will be that the symbol will be **undefined**.

You can use the Project Options in the IDE to create application-wide compiler defines. Please see the Modifying Project Options topic for more information.

Un-Defining Symbols

You can remove a compiler define using the following syntax:

```
{$UNDEF <Symbol>}
```

Testing for Defined Symbols

To test whether a symbol has been defined, you can use the following syntax:

```
{$IFDEF <Symbol>}
// Include this code if the symbol is defined
[{$ELSE}]
// Include this code if the symbol is not defined (optional)
{$ENDIF}
```

To test whether a symbol has **not** been defined, you can use the following syntax:

```
{$IFNDEF <Symbol>}
// Include this code if the symbol is not defined
[{$ELSE}]
// Include this code if the symbol is defined (optional)
{$ENDIF}
```

An IFDEF or IFNDEF test must **always** be terminated with an ENDIF. The ELSE conditional branch is optional.

Built-In Defines

Elevate Web Builder includes several built-in compiler defines that control various aspects of how the component library and applications are built:

Define	Description
DESIGN	Indicates that the code is being compiled for execution in the IDE
BROWSER	Indicates that the code is being compiled for execution in a client application
RUNTIME	Indicates that the code is being compiled for execution in a server application
CLIENT	Indicates that the code is being compiled for client usage
SERVER	Indicates that the code is being compiled for server usage
VERNNN	Indicates the major/minor version of the Elevate Web Builder compiler, where NNN is the major/minor version number (without the period between the major and minor versions) **Note** For a given major verison, a compiler define is also created automatically for any minor versions that belong to the same major version, but are prior to the current minor version. For example, if the current major/minor version is 3.02, then the following compiler defines will be present: VER300, VER301, and VER302

Example

The following is code from the standard component library that tests for the special DESIGN symbol to determine whether to use the WebDesign (IDE run-time) or the WebDOM (browser run-time) unit:

```
{$IFDEF BROWSER}
uses WebDOM;
{$ENDIF}
{$IFDEF SERVER}
uses WebSrvr;
{$ENDIF}
```

8.3 Types

Elevate Web Builder supports most basic Object Pascal types, and these types are detailed below.

Exact Numeric Types

Exact numeric types are used when you wish to store a numeric value in its exact representation without accumulating rounding errors.

Type	Description
Integer	A 52-bit, signed integer value in client applications (JavaScript), and a 64-bit signed integer value in server applications

Exact numeric literals use the minus (-) as the negative sign character, the plus (+) as the positive sign character, and scientific notation is not supported. In addition, hexadecimal literals can be specified by prefacing the hexadecimal value with the dollar sign ($).

The following are examples of exact numeric literals:

```
var
   MyInteger: Integer;
begin
   MyInteger := 100; // Assign 100 to the Integer variable
   MyInteger := $64; // Assign 100 as hexadecimal to the Integer variable
end;
```

Approximate Numeric Types

Approximate numeric types are used when you wish to store a numeric value in an approximate representation with a floating decimal point. Using approximate numeric types can cause rounding errors due to the fact that certain numbers such as 0.33 cannot be accurately represented using floating-point precision.

Type	Description
Double	A 64-bit, floating-point numeric value with a maximum precision of 16 digits.

Approximate numeric literals use the period (.) as the decimal point character, the minus (-) as the negative sign character, the plus (+) as the positive sign character, and scientific notation is supported via E (e or E) as the exponent character followed by a plus (+) or minus (-) character and the actual exponent value.

The following are examples of approximate numeric literals:

```
var
   MyDouble: Double;
begin
   MyDouble := -100.25; // Assign -100.25 to the Double variable
end;
```

String/Character Types

String types are used when you wish to store a character string of any length up to 2GB. String types always use the Unicode character set for the characters that comprise the string. Character types store a single character, and also use the Unicode character set.

Type	Description
String	A string value with a variable number of characters.
Char	A single character.

String literals use the single quote (') character to identify themselves as such. Any single quotes enclosed inside of the literal must be escaped by prefacing them with another single quote. In addition, single character constants may be specified using their literal value or by prefacing their ordinal character set position with the pound sign (#) character. To reference a specific character in a string, use the left and right brackets ([]) with the 1-based integer position of the character being referenced.

Note
Strings are immutable, meaning that they cannot be modified in-place by assigning new character values at specific positions in the string. They must always be copied and then assigned to a new string in order to be modified.

The following are examples of string/character literals:

```
var
   MyString: String;
   MyCharacter: Char;
begin
   MyString := 'This is a test'; // Assign "This is a test"
                                 // to the String variable
   MyString := #13+#10; // Assign a carriage return and
                        // linefeed to the String variable
   MyCharacter := MyString[2]; // Assign the second character
                               // from the String variable to
                               // the Char variable
end;
```

Date/Time Types

Date/time types are used when you wish to store a date, time, or date/time value. Date/time types are actually just integers, so they can be manipulated just like the Integer type. However, you will need to cast the date/time value as an integer in order to use it like an integer.

Type	Description
DateTime	A date/time value containing the number of milliseconds since midnight on January 1, 1970.

Since date/time types are just integers, there isn't any literal representation of a date/time type.

Boolean Types

Boolean types are used to represent the values of True or False.

Type	Description
Boolean	A logical true/false value.

Boolean literals are expressed as the literals True and False (case-insensitive) or 1 and 0 for True and False, respectively.

The following are examples of boolean literals:

```
var
   MyBoolean: Boolean;
begin
   MyBoolean := False; // Assign False to the Boolean variable
end;
```

8.4 Operators

Elevate Web Builder supports most Object Pascal operators, and these operators are detailed below.

Boolean Operators

The following are the boolean operators, ordered by their operator precedence:

Operator	Description
not	Flips a boolean expression so that True becomes False, or vice-versa.
and	Returns True if both the left and right boolean expressions are True.
or	Returns True if either the left or right boolean expression is True.

Comparison Operators

The following are the comparison operators, ordered by their operator precedence:

Operator	Description
=	Returns True if both the left and right expressions are equal.
<>	Returns True if both the left and right expressions are not equal.
>	Returns True if the left expression is greater than the right expression.
>=	Returns True if the left expression is greater than or equal to the right expression.
<	Returns True if the left expression is less than the right expression.
<=	Returns True if the left expression is less than or equal to the right expression.
is	Returns True if the left expression is an instance of the class type specified in the right expression.

Arithmetic Operators

The following are the arithmetic operators, ordered by their operator precedence:

Operator	Description

not	Returns an integer that represents the inverse of all bits in the right integer expression.
or	Returns an integer that represents all set bits in the left and right integer expressions.
xor	Returns an integer that represents all set bits in either the left or right, but not both, integer expressions.
and	Returns an integer that represents all bits that are set in both the left and right integer expressions.
*	Multiplies the left numeric expression by the right numeric expression.
/	Divides the left numeric expression by the right numeric expression.
div	Divides the left integer expression by the right integer expression.
-	Subtracts the right numeric expression from the left numeric expression.
+	Adds the right numeric expression to the left numeric expression.
mod	Returns the remainder derived from dividing the left numeric expression by the right numeric expression.
shl	Returns the left integer expression shifted to the left by the number of bits specified by the right integer expression.
shr	Returns the left integer expression shifted to the right by the number of bits specified by the right integer expression.

String Operators

The following are the string operators, ordered by their operator precedence:

Operator	Description
+	Concatenates the right string expression to the left string expression.

8.5 Statements

Elevate Web Builder supports most Object Pascal statements, and these statements are detailed below. Please see the Function and Procedure Implementations topic for information on how statements are actually used in function and procedure code blocks.

Assignment Statement

```
<Variable> := <Type-Compatible Expression>;
```

The assignment statement uses the assignment operator (:=) to assign a value from a type-compatible expression on right-hand side of the assignment operator to the variable on the left-hand side of the operator.

The following example illustrates the use of the assignment statement:

```
var
   MyInteger: Integer;
   MyString: String;
begin
   MyInteger := (100 * MyIntegerConstant);
   MyString := 'This is a test';
end;
```

If Statement

```
if <Boolean Expression> then
   <Code Block>
[else if <Boolean Expression> then
   <Code Block>]
[else
   <Code Block>];
```

The if statement is used to provide conditional execution based upon one or more Boolean expressions. When any of the Boolean expressions specified in the if or else if clauses evaluates to True, then the block of statements is executed. The else clause is used to specify that the if none of the Boolean expressions evaluate to True, then the statement block specified for the else clause should be executed.

The following example illustrates the use of the if statement:

```
var
   MyBoolean: Boolean;
begin
   MyBoolean:=False;
   if MyBoolean then
      ShowMessage('This will never execute')
   else
      ShowMessage('This will always execute');
end;
```

Case Statement

```
case <Expression> of
   <Expression>[, <Expression}:
      <Code Block>;
   [<Expression>[, <Expression}:
      <Code Block>;]
   [else
      <Code Block>;]
   end;
```

The case statement is used to provide conditional execution based upon one or more expression comparisons. The expression directly after the case clause is compared against each expression specified before the colon (:). If any of the expression comparisons are equal, then the block of statements directly after the colon is executed. The else clause is used to specify that if none of the expression comparisons are equal, then the statement block specified for the else clause should be executed.

The following example illustrates the use of the case statement:

```
var
   MyString: String;
begin
   MyString:='Hello World';
   case MyString of
      'Hello':
         ShowMessage('Hello');
      'World':
         ShowMessage('World');
      else
         ShowMessage('None of the above');
      end;
end;
```

While Statement

```
while <Boolean Expression> do
      <Code Block>;
```

The while statement is used to provide a looping construct based upon a Boolean expression comparison. The Boolean expression directly after the while clause is compared before every execution of the block of statements. If the Boolean expression evaluates to False, then the loop is terminated and execution will continue on the statement after the block of statements that belong to the while statement.

The following example illustrates the use of the while statement:

```
var
   MyBoolean: Boolean;
begin
   MyBoolean:=True;
   while MyBoolean do
      begin
      ShowMessage('Still looping...');
      if MyBoolean then
```

```
         MyBoolean:=False;
      end;
end;
```

Repeat Statement

```
repeat
      <Code Block>;
until <Boolean Expression>;
```

The repeat statement is used to provide a looping construct based upon a Boolean expression comparison. The Boolean expression directly after the util clause is compared after every execution of the block of statements. If the Boolean expression evaluates to True, then the loop is terminated and execution will continue on the statement after the block of statements that belong to the repeat statement.

The following example illustrates the use of the repeat statement:

```
var
   MyBoolean: Boolean;
begin
   MyBoolean:=False;
   repeat
      begin
      ShowMessage('Still looping...');
      if (not MyBoolean) then
         MyBoolean:=True;
      end;
   until MyBoolean;
end;
```

For Statement

```
for <Integer Value Assignment> to|downto <Integer Expression> do
      <Code Block>;
```

The for statement is used to provide a looping construct based upon an incrementing or decrementing integer value comparison. The loop is seeded with an an integer value assignment that is an assignment statement without a semicolon statement terminator (;). If the to clause is used, then the integer value will be incremented by one for every iteration of the loop, and if the downto clause is used, then the integer value will be decremented by one for every interation of the loop. The integer expression after the to or downto clause serves as the terminator for the loop. Once the integer value is equal to the value of the specified integer expression, the loop is terminated and execution will continue on the statement after the block of statements that belong to the for statement.

The following example illustrates the use of the for statement:

```
var
   MyInteger: Integer;
   MyString: String='Hello world';
begin
   for MyInteger:=1 to Length(MyString) do
      ShowMessage('Character is '+MyString[MyInteger]+'...');
```

```
end;
```

Break Statement

```
break;
```

The break statement is used to unconditionally break out of any looping statement (while, repeat, or for). Any time a break statement is encountered, the loop is terminated and execution will continue on the statement after the block of statements that belong to the looping statement.

The following example illustrates the use of the break statement:

```
var
   MyInteger: Integer;
   MyString: String='Hello world';
begin
   for MyInteger:=1 to Length(MyString) do
      begin
      ShowMessage('Character is '+MyString[MyInteger]+'...');
      if MyString[MyInteger]='w' then
         break;
      end;
end;
```

Continue Statement

```
continue;
```

The continue statement is used to unconditionally stop executing any and all remaining statements in the block of statements for a looping statement (while, repeat, or for) and return to the top of the looping statement.

The following example illustrates the use of the continue statement:

```
var
   MyBoolean: Boolean;
begin
   MyBoolean:=True;
   while MyBoolean do
      begin
      ShowMessage('Still looping...forever');
      continue;
      if MyBoolean then
         MyBoolean:=False;
      end;
end;
```

Exit Statement

```
exit;
```

The exit statement is used to unconditionally stop executing any and all remaining statements in a function or procedure, and leave the function or procedure.

With Statement

```
with <Class Instance> do
   <Code Block>;
```

The with statement is used to introduce a class instance into the scope of the block of statements specified after the do clause. This is useful when one needs to reference several different properties or methods of the class instance and would like to avoid repeatedly typing the same class instance reference.

The following example illustrates the use of the with statement:

```
var
   MyInstance: TMyClass;
begin
   MyInstance:=TMyClass.Create;
   with MyInstance do
      begin
      MyIntegerProperty:=100;
      MyStringProperty:='Hello world';
      end;
   MyInstance.Free;
end;
```

Code Blocks

One or more statements in succession is referred to as a code block. If more than one statement is included in a code block, then the code block must be expressed with the **begin** and **end** keywords:

```
begin
 <Statement>;
 [<Statement>];
end;
```

For example, the following for loop can be expressed without the begin and end keywords because its code block only consists of a single statement:

```
var
   MyInteger: Integer;
   MyString: String='Hello world';
   MyOtherString: String='';
begin
   for MyInteger:=1 to Length(MyString) do
      MyOtherString := MyOtherString + MyString[MyInteger];
end;
```

Code blocks can be nested as many levels deep as necessary.

Statement Termination

One of the most confusing aspects of the Object Pascal language is how to decide when to terminate a statement. Normally, all statements must be terminated with the statement terminator character (;) when the statements are used by themselves. For example, the following assignment statement is terminated in a normal fashion because it is used by itself in a code block:

```
var
   MyString: String;
begin
   MyString := 'This is a test';
end;
```

However, when a statement is embedded within another container statement, such as an if statement, the statement terminator may not be needed. For example, the same assignment statement used above would look like this when used in an if statement:

```
var
   MyString: String;
begin
   if MyParameter then
      MyString := 'This is a test'
   else
      MyString := 'This is not a test';
end;
```

The first assignment statement does not require a statement terminator because it is considered part of the if statement, whereas the second assignment statement has a statement terminator because it is used in the else clause of the if statement.

The exception to this rule occurs when a statement is used within a code block. Statements **always** require a statement terminator when used in a code block.

8.6 Units

The Elevate Web Builder language uses source units for all code in an application. Every source unit in an application has the following structure:

```
unit Unit1;

interface  // Unit interface section

implementation  // Unit implementation section

initialization  // Unit initialization section

finalization    // Unit finalization section

end.
```

Every source unit must begin with the keyword **unit** followed by the name of the unit (without file extension) and the statement terminator (;). In addition, every source unit must end with the **end** keyword followed by a period (.).

The interface and implementation section keywords are also required. The initialization and finalization code blocks are both optional, and one can be specified without the other.

Interface and Implementation Sections

The interface and implementation sections are very similar in structure. The main differences are in the scope (visibility) of each section (private or public) and whether the section can only contain declarations and not implementation code:

- All declarations and code in the implementation section are private to the source unit and cannot be referenced by other souce units. All declarations in the interface section are public to both the current source unit and other source units. For example, if you were to declare the TMyClass class in the implementation section of UnitA, then even if UnitB included a UnitA reference in its uses clause (interface or implementation, it doesn't matter), the TMyClass class declaration would still not be "visible" to UnitB. Please see the Scope topic for more information.
- The interface section can only contain declarations, whereas the implementation section can also include implementation code for functions, procedures, and methods of classes (functions and procedures declared in classes).

Both the interface and implementation sections have the following elements in common:

Uses Clause

The uses clause is a comma-separated list of source unit names (*.wbs, but specified without the file extension). This clause tells the compiler which source units are being referenced by the code in the interface or implementation sections.

Const Clause

The const clause is used to declare constants. Please see the Constant Declarations topic for more information on declaring constants. You can declare as many constants as you wish within the same const clause.

Type Clause

The type clause is used to declare types. Please see the Type Declarations topic for more information on declaring

types. You can declare as many types and classes as you wish within the same type clause.

Var Clause

The var clause is used to declare global variables. Please see the Variable Declarations topic for more information on declaring variables. You can declare as many variables as you wish within the same var clause.

Function and Procedure Declarations

You can include function and procedure declarations anywhere in an interface or implementation section. However, normally one would only include a function or procedure declaration in the interface section. Since a function or procedure is actually implemented in the implementation section, there is no purpose to declaring the function or procedure there twice. Please see the Function and Procedure Declarations topic for more information on declaring functions and procedures.

> **Note**
> None of the various section elements are required. In fact, one can have a valid source unit that includes nothing but the unit name, the interface and implementation clauses, and the end keyword. It would be of little use, but it would still be valid.

Initialization and Finalization Sections

The initialization and finalization sections are used to add code blocks for initializing variables in a unit at application startup and for freeing any resources acquired during execution at application shutdown. For example, in the client component library, an instance of the TApplication component is automatically created and freed in the initialization and finalization code blocks of the WebForms unit:

```
initialization
   Application:=TApplication.Create(nil);
finalization
   Application.Free;
   Application:=nil;
end.
```

The order in which the initialization and finalization code blocks are executed is determined by the order of the unit references in the uses clause of the project source file, as well as the order of the unit references in the uses clauses of the interface and implementation sections of each referenced unit.

The initialization order is as follows:

- The units in the project source file's uses clause are initialized in the order in which they are specified.
- The units in each referenced unit's uses clause are initialized in the order in which they are specified. The units in the uses clause in the interface section of the source unit are initialized first, followed by the units in the uses clause in the implementation section of the source unit.
- After all referenced units have been initialized, the initialization code block for the current unit is executed.

The finalization order is the reverse of the above:

- The units in the project source file's uses clause are finalized in the reverse order in which they are specified.
- The finalization code block for the current unit is executed.

- The units in each referenced unit's uses clause are finalized in the reverse order in which they are specified. The units in the uses clause in the implementation section of the source unit are finalized first, followed by the units in the uses clause in the interface section of the source unit.

Project Source File

The project source (.wbp) of an application uses a format similar to a normal unit, but with some key changes:

```
project Project1;

contains Unit1;  // Contains clause

uses WebForms, WebCtrls;  // Uses clause

begin
  // Project source block
end.
```

The key changes are:

- The project source file begins with the **project** keyword instead of the **unit** keyword.
- The project source file has a contains clause. The contains clause is just like a uses clause but also determines which units are considered project units, as opposed to simply units referenced by the project. The IDE uses this information to determine which units should be shown as part of the project in the Project Manager.
- The project source file only contains a single uses clause.
- The project source file does not contain interface, implementation, initialization, and finalization sections. You can add constant, type, class, function/procedure, and variable declarations between the uses clause and the main code block, but this is strictly optional.
- The project source file contains a single code block (begin..end) that is the first code to be executed when the application starts.

8.7 Constant Declarations

Constant declarations must be in the following format:

```
<Constant Declaration>;
[<Constant Declaration>;]

<Constant Declaration> =

   <Constant Name> = <Expression>
```

The <Constant Name> element must follow the rules for identifiers covered in the Introduction topic, and each constant declaration must be terminated with a semicolon statement terminator (;).

The <Expression> element can be any valid expression that does not contain any variable, function, or procedure references. In other words, the expression can only contain other constant references, or literal expressions. Examples of literal expressions can also be found in the Introduction topic.

8.8 Type Declarations

Type declarations must be in the following format:

```
<Type Declaration> | <Function/Procedure Type Declaration> | <Method Type
      Declaration> | <Class Declaration>;
[<Type Declaration> | <Function/Procedure Type Declaration> | <Method Type
      Declaration> | <Class Declaration>;]

<Type Declaration> =

   <Synonym Type Name> = [type] <Type Name>;

<Function/Procedure Type Declaration> =

   <Function/Procedure Type Name> = function|procedure ([<Parameters>])[:
      <Type Name>];

<Method Type Declaration> =

   <Method Type Name> = function|procedure ([<Parameters>])[: <Type Name>] of
      object;

<Class Declaration> =

   <Class Name> = class [(<Ancestor Class Name>)]
      [<Private Class Members>]
      [<Protected Class Members>]
      [<Public Class Members>]
      end;
```

The <Synonym Type Name>, <Type Name>, <Method Type Name>, <Class Name>, and <Ancestor Class Name> elements must follow the rules for identifiers covered in the Introduction topic, and each type or class declaration must be terminated with a semicolon statement terminator (;).

A synonym type declaration is useful for situations where one wants to use a specific name for a generic type such as an Integer, String, or Boolean. For example, in the client component library, the following type declaration can be found in the WebUI source unit:

```
TColor = type Integer;
```

The optional **type** keyword found before the Integer type name above is used to specify that the declared type (TColor) should be considered a unique type and essentially creates a new Integer type called TColor. The IDE uses this information to determine which property editor to show for a property when displaying the properties of a component in the component inspector.

> **Note**
> The synonym type retains type compatibility with the type name that it is associated with. For example, with the above declaration the compiler will still allow you to use an Integer in any expression where a TColor type is required.

A function/procedure type declaration is used to declare a function/procedure reference type. Function/procedure

references point to a function/procedure declaration, and are used to treat functions/procedures as data. The compiler automatically figures out when a function/procedure reference is being assigned to a variable declared with such a reference type, and when a variable that is declared with such a reference type is being used to call the function/procedure reference contained in the variable. You can only assign references to functions/procedures that have the same signature as the target variable's function/procedure reference type. For example, this assignment is not valid:

```
type

   TFuncRef = function (Value: Integer): Integer;  // Returns an integer

implementation

function DoSomething(Value: Integer): Boolean;  // Returns a boolean
begin
   Result:=(Value=100);
end;

procedure DoSomethingElse;
var
   FuncRef: TFuncRef;
begin
   FuncRef:=DoSomething;  //  This will cause a compiler error !!!!
   ShowMessage(IntToStr(FuncRef(100)));
end;
```

A method type declaration is used to declare a method reference type. A method reference type is like a function/procedure reference type, except that it also includes the class instance to be used when calling the method contained within a variable declared with a method reference type. For more information on method reference type declarations, please see the Events topic.

A class declaration can include an ancestor class name, if applicable. If one is declaring a class that inherits from the base TObject class, then the ancestor class name does not need to be specified. For more information on class declarations, please see the Classes topic.

8.9 Variable Declarations

Variable declarations must be in the following format:

```
<Variable Declaration>;
[<Variable Declaration>;]

<Variable Declaration> =

   <Variable Name> [,<Variable Name>]: <Type Name> = <Default Expression>
```

The <Variable Name> and <Type Name> elements must follow the rules for identifiers covered in the Introduction topic, and each variable declaration must be terminated with a semicolon statement terminator (;).

The <Default Expression> element can be any valid expression that does not contain any variable, function, or procedure references. In other words, the expression can only contain other constant references, or literal expressions. Examples of literal expressions can also be found in the Introduction topic. The <Default Expression> element is used to initialize the variable to a specific value. This is useful in ensuring that a variable is not in an uninitialized state when it is referenced in code.

Warning
If you don't set a default expression for a variable declaration and do not assign a value to the variable, then the value of the variable is undetermined. You should always make sure that such global variable declarations are initialized properly through a default expression or assignment. This does not, however, apply to variables declared within a class. Please see the Variables topic for more information on declaring variables in a class.

8.10 Function and Procedure Declarations

Function and procedure declarations must be in the following format:

```
<Function Declaration> | <Procedure Declaration>;
[<Function Declaration> | <Procedure Declaration>;]

<Function Declaration> =

   function <Function Name> [<Parameters>]: <Type Name>

<Procedure Declaration> =

   procedure <Procedure Name> [<Parameters>]

<Parameters> =

  (<Parameter Declaration>[; <Parameter Declaration>])

<Parameter Declaration> =

   [const] <Parameter Name> [,<Parameter Name>]: <Type Name>
```

The <Function Name>, <Procedure Name>, <Parameter Name>, and <Type Name> elements must follow the rules for identifiers covered in the Introduction topic, and each function or procedure declaration must be terminated with a semicolon statement terminator (;).

The only difference between a function and procedure is that a function returns a result value, whereas a procedure does not. This means that a function can be used in an expression like a variable or constant, but a procedure can only be used like a statement.

Please see the Function and Procedure Implementations topic for more information on implementing functions and procedures.

8.11 Function and Procedure Implementations

Function and procedure implementations are the executable code in a unit. The implementations of functions/procedures are added to the implementation section of a unit and consist of a variable declaration block, if necessary, followed by a code block:

```
procedure MyProcedure;
var
   MyVariable: String;
   MyOtherVariable: Integer;
begin
   // Code block
end;
```

Note
The implementation of a function/procedure repeats the same declaration of the function or procedure name followed by the parameters (if present) and return type (for functions). If the implementation does not use the exact same declaration, then the compiler will issue an error.

The variable declarations follow the same declaration rules as the **var** clause in a unit. Please see the Variable Declarations topic for more information.

A code block consists of a **begin** keyword followed by a series of statements and an **end** keyword. The code block that is used in a function or procedure implementation is always terminated with a statement terminator (;). However, code blocks can be nested within other statements, such as conditional if statements. In such cases, please refer to the documentation on the statement to determine if a statement terminator is necessary after the end keyword at the end of a code block.

Returning Results From a Function

Procedures do not return a result, whereas functions do. Every function has an implicit **Result** variable that can be assigned a value that is returned as the result of the function:

```
function MyFunction(const MyParameter: String): String;
begin
   Result := 'The parameter value is ' + MyParameter;
end;
```

The type of the Result variable is determined by the type declaration for the function's return value.

8.12 Enumerations

An enumeration is a collection of symbols used to represent a specific set of values. An enumeration must be declared as a specific type and, because they are typed, enumerations offer the additional benefit of preventing improper symbolic values that aren't part of the enumeration from being used anywhere that the enumeration type is required.

An enumeration is declared as follows:

```
<EnumerationName> = (<Member Name>[,<Member Name>]);
```

For example, the following enumeration type declaration is from the WebUI unit in the client component library and specifies the various cursor types that can be used in a UI element:

```
TCursor = (crAuto,crCrossHair,crDefault,crHelp,crMove,crPointer,
           crProgress,crSizeNESW,crSizeNS,crSizeNWSE,crSizeWE,
           crText,crWait);
```

Note
Internally, enumerations are handled as integers by the compiler, and you can cast enumerations as integers and integers as enumerations.

8.13 Arrays

An array is a collection of values, all of the same type. The values in an array are referred to as the array's elements. Arrays are dynamic, meaning that their length is not specified during their declaration and can be increased or decreased at run-time, as necessary.

Array Declarations

Arrays are declared by prefacing the type name of the array with the keywords **array of**:

```
array of <Type Name>
```

For example, to declare a Boolean array variable, one would use the following declaration:

```
var
   MyBooleanArray: array of Boolean;
```

Please see the Variable Declarations topic for more information on declaring variables.

Getting and Setting the Length of an Array

To get the length of an array, use the Length function. Likewise, use the SetLength function to set the length of an array:

```
var
   MyArray: array of Integer;
begin
   SetLength(MyArray,10);
   ShowMessage(Length(MyArray)); // Displays the value 10
end;
```

Referencing an Array Element

Each element in an array can be accessed via its 0-based ordinal position. Square brackets directly after the array variable or parameter name are used to reference an element in an array. For example, to access the 3rd element in an array, one would use the following construct:

```
MyArray[2]
```

Warning
If you try to access an element that does not exist because it is beyond the length of the array, you will cause a run-time error. Also, you must declare an array variable with a default value (see below), or use the SetLength function to set the length of an array, before attempting to reference any of the array elements. Failure to do so will result in a run-time error.

Array Constants

Just like any other variable, array variables can be initialized to a default value by specifiying an array constant as the default value in the variable declaration. This is done by enclosing a comma-delimited list of array elements in square brackets ([]):

```
[<Element> ,<Element>...]
```

In the following example the MyBooleanArray variable will be initialized to an array of Boolean elements that has a length of 3 and consists of elements that are True, False, and True, respectively:

```
var
   MyBooleanArray: array of Boolean = [True, False, True];
```

In addition to variable defaults, array constants can be used to pass array values to the parameters of functions or procedures. For example, if we want to call a procedure declared as follows:

```
function ListStrings(Value: array of String): String;
var
   I: Integer;
begin
   Result := '';
   for I := 0 to Length(Value) - 1 do
      begin
      if (I > 0) then
         Result := Result + ', ';
      Result := Result + Value[I];
      end;
end;
```

We could do so by simply passing a constant array to it, as follows:

```
ShowMessage(ListStrings(['These','are','some','words']);
```

Note
The values specified for the elements in an array constant must be type-compatible with the declared type of the array.

8.14 Classes

The Object Pascal language used by Elevate Web Builder is an object-oriented language that allows one to define classes that represent objects with their own data (variables and properties) and behaviors (functions and procedures).

> **Note**
> Functions and procedures that are declared in a class are referred to as "methods". For the rest of this topic the term "methods" will be used to represent the functions and procedures declared in a class.

Classes are useful because they offer:

- **Encapsulation**: Both data and behaviors are combined into one logical construct, and data that is internal to the class can be hidden so that only the class itself can access it. Also, properties can be defined that offer a specific interface to the data contained in a class, thus avoiding exposing internal data directly or requiring that all access to the data be done through methods.
- **Inheritance**: Classes can descend from other classes and inherit the functionality of the ancestor class in the process, thus forming what is termed a "class hierarchy". There is no limit to the depth of such a hierarchy. In addition, the functionality of ancestor class(es) can be overridden in descendant classes in order to supplement or completely replace the base functionality.

> **Note**
> Elevate Web Builder only supports single inheritance. This means that each class can only descend from one class, and there is only one single path between an ancestor class and a descendant class.

Class Declarations

Before a class can be used, it must be declared in the type section of a unit. A class declaration consists of the following:

```
<Class Name> = class [(<Ancestor Class Name>)]
   [<Private Class Members>]
   [<Protected Class Members>]
   [<Public Class Members>]
   [<Published Class Members>]
   end;
```

> **Note**
> To keep with the traditional coding style of Object Pascal, all <Class Name> specifications should normally begin with a "T" prefix (stands for "Type").

If you do not specify an <Ancestor Class Name>, then the class will inherit from the system-defined TObject class (see below).

The private, protected, public, and published designations specify the scope of the class members. The scope of the class members determine their visibility to descendant class declarations, as well as any code that uses an instance of the class. In addition, the published scope determines which properties are visible at design-time in the IDE and

which properties are streamable using the persistence functionality in the component library. Please see the Scope topic for more information on the various class member scope designations.

TObject

The TObject class is the base ancestor class for all classes. The declaration for the TObject class is as follows:

```
TObject = class
   public
      constructor Create; virtual;
      destructor Destroy; virtual;
      class procedure Free;
      class function ClassType: TClass;
      class function ClassName: String;
      class function ClassParent: TClass;
   end;
```

The Create method is the class constructor and the Destroy method is the class destructor. The constructor method is called when a class instance is created, and the destructor method is called when a class instance is freed. Please see the Methods topic for more information on constructors and destructors.

The ClassType method is a class method that returns the class type as a result. This method is useful in situations where you need to interrogate the class type of either a class or a class instance.

The ClassName method is a class method that returns the class name as a result. The client component library uses this method a lot in order to link control classes to specific control interfaces.

The ClassParent method is a class method that returns the parent class type as a result. This information can be used to determine the ancestry of a class.

Note
The above Class* methods are class methods, which means that they are static methods that can operate on both classes and class instances. Please see the Methods topic for more information on class methods.

Class Members

The class members in a class declaration are the various variables, properties, methods, and events that define the data and behaviors of the class. Class members can be declared in any order within a specific scope in a class declaration.

Please use the following links to get more information on each type of class member:

Variables
Methods
Properties
Events

8.15 Variables (In Classes)

Variables are declared in a class just like they are declared in a unit or function/procedure. Please see the Variable Declarations topic for more information.

> **Note**
> As a code convention, variable declarations in classes are normally prefaced with an "F" to distinguish them from other variables and properties. The "F" stands for "Field", but this manual will refer to them as variables and not "fields".

If you don't set a default expression for a variable declaration in a class declaration, then the variable will be automatically initialized to the appropriate value for the type when an instance of the class is created:

Type	Initial Value
String	''
Char	#0
Integer Double	0
Enumerated Type	Lowest Member Value
Boolean	False
DateTime	0
Class Type Class Instance Array Method Reference Function/Procedure Reference	nil

Class Variables

Class variables are special types of variables that are sometimes referred to as "static" variables. They can be referenced from methods in class instances and from class methods in class types, and are useful for storing data that doesn't change between instances of a class. Please see the Methods and Properties topics for more information on class methods and properties.

Class variables are declared by prefacing a variable declaration with the **class** keyword. For example, the following class declaration includes a class variable that keeps track of how many instances of the class have been created:

```
TMyClass = class
   private
      class FCreateCount: Integer;
   public
      constructor Create; override;
      class property CreateCount: Integer read FCreateCount;
   end;

implementation

constructor TMyClass.Create;
begin
```

```
   inherited Create;
   Inc(FCreateCount);
end;
```

The CreateCount class property above, and subsequently the FCreateCount class variable, could be accessed in two different ways. The first way is by referring to the CreateCount class property for an instance of the TMyClass class:

```
procedure ShowCreateCount;
var
   TempInstance: TMyClass;
begin
   TempInstance:=TMyClass.Create;
   try
      ShowMessage(IntToStr(TempInstance.CreateCount));
   finally
      TempInstance.Free;
   end;
end;
```

The second way is by using a direct class reference:

```
procedure ShowCreateCount;
begin
   ShowMessage(IntToStr(TMyClass.CreateCount));
end;
```

The second way is easiest when you don't have an instance of the class available.

One of the most significant benefits of class variables is that only one instance of each class variable ever exists, thus saving memory. They are also very useful for implementing singleton instances of classes and implementing namespaces for code that otherwise would use normal functions and procedures declared outside of a class.

8.16 Methods

Methods are simply functions and procedures that are declared as part of a class declaration. They are declared in the same way as functions and procedures that are declared in a unit. Please see the Function and Procedure Declarations topic for more information on the proper syntax. However, methods offer three additional keywords: **virtual**, **abstract**, and **override**.

Virtual Methods

The **virtual** keyword allows you to specify whether or not a method can be overridden by descendant classes. Virtual methods form the basis of inheritance in an object-oriented language because they allow the developer to supplement or replace existing functionality in an ancestor class with functionality that is more specific to the current class. For example, consider the following example class declarations and implementations:

```
interface

   TVehicle = class
      protected
         function GetNumWheels: Integer; virtual;
      public
         property NumWheels: Integer read GetNumWheels;
      end;

   TTruck = class(TVehicle)
      protected
         function GetNumWheels: Integer; override;
      end;

implementation

function TVehicle.GetNumWheels: Integer;
begin
   Result := 4;
end;

function TTruck.GetNumWheels: Integer;
begin
   Result := 10;
end;
```

As you can see, the default value returned from the GetNumWheels method is 4. But, because the GetNumWheels method is virtual, descendant classes like the TTruck class can override the method to provide a different result that is accurate for the type of vehicle being represented by the class.

Note
A virtual method does not have to be present in the immediate ancestor class in order for it to be overridden. You can override **any** virtual method that exists in **any** ancestor class, no matter how far removed it is from the class being declared. Also, once a method is declared as virtual, it is always virtual and capable of being overridden by a descendant class.

In addition, you can use the **abstract** keyword to specify that the virtual method isn't actually implemented in the current class, but rather **must** be implemented by descendant classes. Using the above example, the base TVehicle class could be declared and implemented as follows instead:

```
interface

   TVehicle = class
      protected
         function GetNumWheels: Integer; virtual; abstract;
      public
         property NumWheels: Integer read GetNumWheels;
      end;

   TTruck = class(TVehicle)
      protected
         function GetNumWheels: Integer; override;
      end;

implementation

function TTruck.GetNumWheels: Integer;
begin
   Result := 10;
end;
```

Note
Any classes that contain abstract methods cannot be directly created. Any attempt to do so will cause a run-time error.

If you want to augment, or add to, the functionality present in the virtual method of an ancestor class, then you can use the **inherited** keyword in the implementation of the descendant class method to do so. For example, in this class hierarchy the TCar class defines a GetAvailableColors method that returns a comma-delimited list of the default colors that a car is available in for the car manufacturer. The descendant TJaguar class that represents a sports car overrides the GetAvailableColors method to specify additional colors that the sports car is available in:

```
interface

   TCar = class
      protected
         function GetAvailableColors: String; virtual;
      public
         property AvailableColors: String read GetAvailableColors;
      end;

   TJaguar = class(TCar)
      protected
         function GetAvailableColors: String; override;
      end;

implementation

function TCar.GetAvailableColors: String;
begin
   Result := 'Red, Black, White';
end;

function TJaguar.GetAvailableColors: String;
begin
   Result := inherited GetAvailableColors + ', Silver';
end;
```

Overloaded Methods

Overloaded methods are methods that have more than one declaration in a class and each declaration has the same name but different parameters. This is very useful for situations where a method may need to be called using different parameters. For example, consider the following class declaration:

```
TCustomers = class
   public
      procedure Delete(ID: Integer);
      procedure Delete(const Name: String);
   end;
```

In this example, the TCustomers class has overloaded the Delete method so that it can be called with either an integer customer ID or a string customer name.

Although default parameters are usually easier, overloaded methods can also be used to implement optional parameters. For example, the following class allows its Add method to be called with an ID, a name, or both:

```
TCustomers = class
   public
      procedure Add(ID: Integer);
      procedure Add(ID: Integer; const Name: String);
      procedure Add(const Name: String);
   end;
```

Note
Other variants of Object Pascal require that you use the **overload** keyword to indicate that a method is overloaded. Elevate Web Builder does not use the overloaded keyword because it is unnecessary. The compiler knows if two methods have the same declaration, and will issue an error if they do. The compiler also knows how to find the proper method declaration, or whether one exists at all, by how the method is called. Finally, if an overloaded method has one or more declared versions that aren't actually called, the compiler knows this and will not emit the method during compilation.

Class Methods

Class methods are special types of methods that are sometimes referred to as "static" methods. They are callable from class instances and also directly from class references where no instance of the class exists, and are useful for implementing functions and procedures that should be encapsulated within the context of a class, but don't need to access class instance variables, properties, or methods. Class methods can, however, access any class variables or class properties that are also declared in the same class. Please see the Variables and Properties topics for more information on class variables and properties.

Class methods are declared by prefacing a method declaration with the **class** keyword. For example, the following class declaration includes a class variable that keeps track of how many instances of the class have been created along with a class method that returns the creation count:

```
TMyClass = class
   private
      class FCreateCount: Integer;
   public
```

```
      constructor Create; override;
      class function GetCreateCount: Integer;
   end;

implementation

constructor TMyClass.Create;
begin
   inherited Create;
   Inc(FCreateCount);
end;
```

The GetCreateCount class method above could be accessed in two different ways. The first way is by referring to the GetCreateCount class method for an instance of the TMyClass class:

```
procedure ShowCreateCount;
var
   TempInstance: TMyClass;
begin
   TempInstance:=TMyClass.Create;
   try
      ShowMessage(IntToStr(TempInstance.GetCreateCount));
   finally
      TempInstance.Free;
   end;
end;
```

The second way is by using a direct class reference:

```
procedure ShowCreateCount;
begin
   ShowMessage(IntToStr(TMyClass.GetCreateCount));
end;
```

The second way is easiest when you don't have an instance of the class available.

Class methods are very useful for implementing factory classes that create instances, implementing singleton instances of classes, and implementing namespaces for code that otherwise would use normal functions and procedures declared outside of a class.

Constructors and Destructors

Constructors and destructors are special methods that handle the process of creating a class instance and destroying it. These methods are crucial to ensuring that resources are properly allocated and initialized during the creation of class instances, and that resources are properly disposed of during the destruction of class instances. The base TObject class declaration contains both a constructor called **Create** and a destructor called **Destroy**.

> **Note**
> Constructors and destructors are completely optional. If a class declaration doesn't contain a constructor and/or destructor, then the ancestor class's constructor and/or destructor is used instead. If the ancestor class doesn't contain a constructor and/or destructor, then it's ancestor class is used and so on, until the base TObject class is reached by the compiler.

Constructors

Constructors are declared by prefacing a method declaration with the keyword **constructor**. Constructors must be declared as a procedure called **Create** with no result type declaration due to the fact the result is implicitly an instance of the class in which the declaration exists. Trying to declare a constructor with a different name or with a result type will cause a compiler error. If the declared constructor does not accept any parameters, then it must also be declared as an **override** of the base TObject Create constructor. Constructors can be overloaded, so it is possible to declare different constructors with different parameters.

> **Warning**
> Do not call constructors on instances of classes. Constructors are class, or static, methods, and should only be called on a class type itself in order to create an instance of the class.

The following is an example of a class that declares both an override of the base TObject Create constructor, as well as creates its own overloaded constructor:

```
interface

   TCustomer = class
      private
         FID: Integer;
         FName: String;
      public
         property ID: Integer read FID write FID;
         property Name: String read FName write FName;
      end;

   TCustomers = class
      private
         FCustomers: TObjectList;  // TObjectList class is declared
                                   // in the WebCore unit
         procedure CreateDemoCustomers(Value: Integer);
         function GetNumCustomers: Integer;
         function GetCustomer(ID: Integer): TCustomer;
         function GetCustomer(const Name: String): TCustomer;
      public
         constructor Create; override;
         constructor Create(NumDemoCustomers: Integer);
         property NumCustomers: Integer read GetNumCustomers;
         property Customer[ID: Integer]: TCustomer read GetCustomer; default;
         property Customer[const Name: String]: TCustomer read GetCustomer;
      default;
      end;

implementation

constructor TCustomers.Create;
begin
   inherited Create;
   FCustomers:=TObjectList.Create;
end;

constructor TCustomers.Create(NumDemoCustomers: Integer);
begin
   Create;
   CreateCustomers(NumDemoCustomers);
end;
```

```
procedure TCustomers.CreateDemoCustomers(Value: Integer);
var
   I: Integer;
   TempCustomer: TCustomer;
begin
   for I:=0 to Value-1 do
      begin
      TempCustomer:=TCustomer.Create;
      with TempCustomer do
         begin
         ID:=I;
         Name:='Demo Customer #'+IntToStr(I);
         end;
      FCustomers.Add(TempCustomer);
      end;
end;

function TCustomers.GetNumCustomers: Integer;
begin
   Result:=FCustomers.Count;
end;

function TCustomers.GetCustomer(ID: Integer): TCustomer;
var
   I: Integer;
begin
   Result:=nil;
   for I:=0 to FCustomers.Count-1 do
      begin
      if (TCustomer(FCustomers[I]).ID=ID) then
         begin
         Result:=TCustomer(FCustomers[I]);
         Break;
         end;
      end;
end;

function TCustomers.GetCustomer(const Name: String): TCustomer;
var
   I: Integer;
begin
   Result:=nil;
   for I:=0 to FCustomers.Count-1 do
      begin
      if SameStr(TCustomer(FCustomers[I]).Name,Name) then
         begin
         Result:=TCustomer(FCustomers[I]);
         Break;
         end;
      end;
end;
```

Note
The above class declaration includes default array properties. Please see the Properties topic for more information on default array properties.

Destructors

A destructor is declared by prefacing a method declaration with the keyword **destructor**. There can be only one destructor per class and it must be declared as a procedure called **Destroy** that overrides the base TObject Destroy

destructor and has no parameters. Trying to declare a destructor with a different name or with parameters will cause a compiler error. Finally, the destructor must be declared in the **public** scope of the class declaration and cannot be declared in any other scope.

The above example for constructors is an example of a class with an overridden Destroy destructor.

Free Method

Do not call the Destroy method above directly, use the TObject Free method instead. The Free method performs an extra crucial step that the Destroy method does not: the Free method checks to see if the calling instance variable is already nil, and only calls the Destroy method if the instance variable is not nil.

> **Note**
> The only exception to the above rule is when calling the inherited Destroy method from within an ancestor class's Destroy method. That is a perfectly valid way to call the Destroy method directly.

Self

Use the special Self keyword in order to reference the current class instance from within a method. This is useful for situations where local variable or parameter names may conflict with a specific variable, method, or property name of the class in which the method resides, and so those identifiers need to be prefixed with the Self keyword.

8.17 Properties

Properties are one of the fundamental ways that Object Pascal provides encapsulation in classes. You can have properties directly reference variables or you can also use methods to control the reading and/or writing of variables. This helps to further hide the implementation details of a class and provide an easy-to-use class interface.

A property is declared as follows:

```
property <Property Name>: <Type Name> read <Variable Name>|<Method Name>
                                      [write <Variable Name>|<Method Name>]
                                      [default <Default Expression>]
                                      [description <Description>]
                                      [;default];
```

All properties must specify a variable or method name in the **read** clause, but the **write** clause is optional. If the write clause is not specified, then the property is implicitly read-only and cannot be assigned a value.

The following example shows a simple class with two read/write property declarations that directly reference variables:

```
interface

   TCustomer = class
      private
         FID: Integer;
         FName: String;
      public
         property ID: Integer read FID write FID;
         property Name: String read FName write FName;
      end;
```

The following example shows the same class, but modified to track modifications to any of the variables:

```
interface

   TCustomer = class
      private
         FID: Integer;
         FName: String;
         FModified: Boolean;
         procedure SetID(Value: Integer);
         procedure SetName(const Value: String);
      public
         property ID: Integer read FID write SetID;
         property Name: String read FName write SetName;
         property Modified: Boolean read FModified;
      end;

implementation

procedure TCustomer.SetID(Value: Integer);
begin
   if (Value <> FID) then
      begin
```

```
      FID:=Value;
      FModified:=True;
      end;
end;

procedure TCustomer.SetName(const Value: String);
begin
   if (not SameStr(Value,FID)) then
      begin
      FName:=Value;
      FModified:=True;
      end;
end;
```

The default clause specifies the default value for a property, and is optional. This default value is used to determine if the property should be streamed or not. If a default value is not provided for a property, and the property is published, then it will always be streamed when an instance of the owner class is streamed.

The description clause specifies the description for a property, and is also optional. This description appears in the IDE's component inspector when the property is declared in the published scope, or is promoted to the published scope in an ancestor class.

The terminating default clause is different from the default value clause above. It is used to specify default array properties and default event properties. See below for more information on default array properties. A default event property is used by the IDE to determine which event handler should automatically be created when a developer double-clicks on a control on the designer surface.

Array Properties

There is one special type of property that is a fairly powerful construct, and that is the array property. An array property is declared as follows:

```
property <Property Name>[[const] <Parameter Name>: <Type Name>]: <Type Name>
              read <Array Variable Name>|<Method Name>
              [write <Array Variable Name>|<Method Name>]; [default;]
```

An array property acts like an array but does not necessarily use an array variable for the read and/or write clauses of the property. Such a property can use methods instead of array variables, with the read clause requiring a method that accepts an identical single parameter that matches the array property parameter declaration and a write clause requiring a method that accepts the same parameter name and type and an additional parameter that can have any name that you wish, but whose type must match the type of the property.

> **Note**
> If you specify an array variable in the read or write clause, then the array property parameter must be an Integer type to reflect the ordinal index into the array.

For example, the following read-only property declares an array property that uses a method to return a specific class instance from an internal list:

```
TCustomers = class
   private
      FCustomers: TObjectList;
      function GetCustomer(ID: Integer): TCustomer;
```

```
   public
      property Customer[ID: Integer]: TCustomer read GetCustomer;
   end;
```

You would reference the Customer array property as follows:

```
var
   TempCustomers: TCustomers;
begin
   TempCustomers:=TCustomers.Create(10);
   try
      ShowMessage(TempCustomers.Customer[2].Name);
   finally
      TempCustomers.Free;
   end;
end;
```

Array properties make it very easy to hide the internal implementation of lists.

Default Array Properties

The above array property example illustrates a common problem with array properties used with classes that encapsulate lists: they tend to bloat the code by requiring the name of the array property to be specified. This is where default array properties are very useful. A default array property has the same declaration as a normal array property, but includes the **default** keyword after the declaration. To continue with the above example, let's change it to a default array property:

```
TCustomers = class
   private
      FCustomers: TObjectList;
      function GetCustomer(ID: Integer): TCustomer;
   public
      property Customer[ID: Integer]: TCustomer read GetCustomer; default;
   end;
```

You would reference the Customer default array property as follows:

```
var
   TempCustomers: TCustomers;
begin
   TempCustomers:=TCustomers.Create(10);
   try
      ShowMessage(TempCustomers[2].Name);
   finally
      TempCustomers.Free;
   end;
end;
```

Notice that since the Customer property is marked as the default array property, you no longer need to specify its name, only the instance variable name of the containing Customers class.

Overloaded Array Properties

One final interesting feature of array properties is that they can be overloaded so that you can have multiple such properties with the same name, but with different declarations. This is especially useful when you want to allow access to a list via different search types. For example, here is the above example with the Customer default array property overloaded with an additional declaration that uses a name parameter for retrieving a customer instead of an ID:

```
TCustomers = class
   private
      FCustomers: TObjectList;
      function GetCustomer(ID: Integer): TCustomer;
      function GetCustomer(const Name: String): TCustomer;
   public
      property Customer[ID: Integer]: TCustomer read GetCustomer; default;
      property Customer[const Name: String]: TCustomer read GetCustomer;
      default;
   end;
```

You could then reference the Customer default array property in both ways:

```
var
   TempCustomers: TCustomers;
begin
   TempCustomers:=TCustomers.Create(10);
   try
      ShowMessage(TempCustomers[2].Name);
      ShowMessage(IntToStr(TempCustomers['Demo Customer #2'].ID));
   finally
      TempCustomers.Free;
   end;
end;
```

Class Properties

Class properties are special types of properties that are sometimes referred to as "static" properties. They can be referenced from class instances and also directly from class references where no instance of the class exists, and are useful for implementing properties that should be encapsulated within the context of a class, but don't need to access class instance variables, properties, or methods. Class properties can, however, access any class variables or class methods that are also declared in the same class. Please see the Variables and Methods topics for more information on class variables and methods.

Class properties are declared by prefacing a property declaration with the **class** keyword. For example, the following class declaration includes a class variable that keeps track of how many instances of the class have been created along with a class property that returns the creation count:

```
TMyClass = class
   private
      class FCreateCount: Integer;
   public
      constructor Create; override;
      class property CreateCount: Integer read FCreateCount;
   end;

implementation

constructor TMyClass.Create;
```

```
begin
   inherited Create;
   Inc(FCreateCount);
end;
```

The CreateCount class property above could be accessed in two different ways. The first way is by referring to the CreateCount class property for an instance of the TMyClass class:

```
procedure ShowCreateCount;
var
   TempInstance: TMyClass;
begin
   TempInstance:=TMyClass.Create;
   try
      ShowMessage(IntToStr(TempInstance.CreateCount));
   finally
      TempInstance.Free;
   end;
end;
```

The second way is by using a direct class reference:

```
procedure ShowCreateCount;
begin
   ShowMessage(IntToStr(TMyClass.CreateCount));
end;
```

The second way is easiest when you don't have an instance of the class available.

8.18 Events

Events are declared just like properties in classes, but read/write a method reference instead of a string, integer, etc. value. A method reference is similar to a class instance reference. But, instead of referring to only a class instance, a method reference contains both a class instance reference and a reference to a method declaration. Method references are useful because they can be called just like a normal method, but can also be assigned to variables and passed as parameters to other methods, procedures, or functions. This allows you to assign specific **behaviors** to the event properties of a class, swap such behaviors in and out with other behaviors, or assign no behavior at all.

Before an event can be declared, the event's method reference type must be declared. A method reference type is declared as follows:

```
<Type Name> = function/procedure ([<Parameters>])[: Type Name>] of object;
```

The procedure or function declaration is identical to a normal procedure or function prototype, except that a procedure or function name is not specified.

For example, consider the following method reference type and class/event declarations:

```
interface

   TStartEvent = procedure (StartingVehicle: TVehicle) of object;

   TVehicle = class
      private
         FOnStart: TStartEvent;
      public
         property OnStart: TStartEvent read FOnStart write FOnStart;
         procedure Start;
      end;

implementation

procedure TVehicle.Start;
begin
   if Assigned(FOnStart) then
      FOnStart(Self);
end;
```

Defining Event Handlers

Once an event is declared as a property in a class, it is still not very useful until the event is assigned an actual method reference. This type of method reference is referred to as an event handler. For example, the following code creates an instance of the TVehicle class and assigns an OnStart event handler:

```
interface

   TGarage = class
      private
         FVehicle: TVehicle;
      protected
         procedure DoVehicleStart(StartingVehicle: TVehicle);
      public
```

```
         constructor Create; override;
         destructor Destroy; override;
      end;

implementation

constructor TGarage.Create;
begin
   inherited Create;
   FVehicle:=TVehicle.Create;
   FVehicle.OnStart:=DoVehicleStart;
   FVehicle.Start;
end;

destructor TGarage.Destroy;
begin
   FVehicle.Free;
   inherited Destroy;
end;

procedure TGarage.DoVehicleStart(StartingVehicle: TVehicle);
begin
   ShowMessage('Vehicle has been started');
end;
```

Note
Event handlers are always called using a reference to the class instance that contains the event handler. So, in the above example when the DoVehicleStart event handler is called, it will be called using the TGarage class instance that was in scope when the OnStart property was assigned.

Method reference assignments and parameters must be type-compatible with the declared type of the target variable or parameter. To illustrate this concept, suppose that the above DoVehicleStart event handler was declared as follows:

```
TGarage = class
   private
      FVehicle: TVehicle;
   protected
      procedure DoVehicleStart;
   public
      constructor Create; override;
      destructor Destroy; override;
   end;
```

This would result in a compiler error on the source line where the event handler is assigned:

```
constructor TGarage.Create;
begin
   inherited Create;
   FVehicle:=TVehicle.Create;
   FVehicle.OnStart:=DoVehicleStart;  // Compiler error here !!!
   FVehicle.Start;
end;
```

The reason for the compiler error is simple: the TStartEvent type of the OnStart event is declared with a TVehicle parameter, and the DoVehicleStart method is not declared with any parameters.

Clearing Event Handlers

Just as you can attach a behavior to the event property of a class in the form of an event handler, you can also remove that behavior by assigning nil to the event property:

```
constructor TGarage.Create;
begin
   inherited Create;
   FVehicle:=TVehicle.Create;
   FVehicle.OnStart:=DoVehicleStart;
   FVehicle.Start;
   FVehicle.OnStart:=nil; // Clear event handler
end;
```

Warning
Because the method reference variables that are used with event properties can be nil, you should always check for this before trying to call such method reference variables. The best way to do so is by using the Assigned function, as illustrated in the TVehicle.Start method implementation:

```
procedure TVehicle.Start;
begin
   if Assigned(FOnStart) then
      FOnStart(Self);
end;
```

Default Events

Components and controls that will be used in the component library can have a default event property. A default event property is the event handler that will be created if the user double-clicks on a component or control in the designer in the IDE. For example, the following TVehicle class has been slightly modified from the above example. It is now a descendant of the TComponent class so that it can be installed into the component library, and now defines the OnStart event as the default event in the published section of the class:

```
interface

   TStartEvent = procedure (StartingVehicle: TVehicle) of object;

   TVehicle = class(TComponent)
      private
         FOnStart: TStartEvent;
      public
         procedure Start;
      published
         property OnStart: TStartEvent read FOnStart write FOnStart; default;
      end;

implementation

procedure TVehicle.Start;
```

```
begin
   if Assigned(FOnStart) then
      FOnStart(Self);
end;
```

A default event is only used in the designer if the event property is published. Please see the Scope topic for more information on published properties.

8.19 Scope

The scope (visibility) of a constant, type/class, or function/procedure declaration is determined by where it appears in a unit, class, or function/procedure.

Unit Scope

The scope of declarations in a unit are determined by whether they are declared in the interface or implementation section of a unit:

- Any declaration in the interface section of a unit is visible to all other declarations or function/procedure implementations in either the interface or implementation sections of the same unit, as well as being visible to the same sections in any units that reference the unit. The interface section of a unit is essentially "public" to everything.
- Any declaration in the implementation section of a unit is visible to all other declarations in the same implementation section only.

```
unit Unit1;

interface

// Public to this unit and all referencing units

implementation

// Private to this unit

end.
```

Function/Procedure Implementation Scope

The implementations of functions and procedures only have one level of scope, and that is the scope of any parameters or local variables, as well as the special Result variable for functions. Parameters and local variables can only be referenced within the function/procedure in which they are declared.

```
// Parameters are private to the function

function MyFunction(const MyParameter: String): String;
var
   MyVariable: String;  // Variables are private to the function
begin

end;
```

Class methods (functions or procedures declared as part of a class) add an additional level of scope that is evaluated after any local variables or parameters. Class methods can access any member variables, properties, or methods that are declared anywhere within the same class in which the referencing method is declared. Class methods can also access any protected, public, or published member variables, properties, or methods that are declared within any ancestor classes of the class in which the referencing method is declared.

Note
In order to specifically reference the current instance of a class from within a class method, use the special **Self** keyword.

See below for more information about the various levels of scope (private, protected, public, published) within a class declaration:

```
interface

type

   TClassA = class
      private
         FMemberVariable: String;
      protected
         procedure DoSomethingMethod;
      public
         property MemberProperty: String read FMemberVariable;
      end;

   TClassB = class(TClassA)
      public
         procedure DoSomethingElseMethod;
      end;

implementation

{ TClassA Implementation }

procedure TClassA.DoSomethingMethod;
begin
   FMemberVariable := 'Test';  // Can access this variable because it is also
                               // declared within the TClassA declaration
end;

{ TClassB Implementation }

procedure TClassB.DoSomethingElseMethod;
begin
   DoSomethingMethod;  // Can access this protected method because it is
                       // declared within the ancestor TClassA declaration
end;
```

There is one exception to the normal scoping rules of a function/procedure, and that is the with statement. The with statement can introduce an object instance as a new level of scope that overrides the normal scope of any local variables, parameters, or class variables/properties/methods.

Class Scope

A class declaration can have up to four different levels of scope for any member variables, properties, or methods:

Scope	Description

Private	Any member variables, properties, or methods declared in this scope are only visible to the properties or methods of the same class declaration.
Protected	Any member variables, properties, or methods declared in this scope are only visible to the properties or methods of the same class declaration, or any descendant class declarations.
Public	Any member variables, properties, or methods declared in this scope are visible everywhere, subject to the scoping rules of the referencing declaration or code block.
Published	Same as Public, but in addition, all properties declared in this scope are streamable and will appear in the IDE's component inspector.

Naming Conflicts and Scope

In certain cases, the scoping rules are used by the compiler to resolve naming conflicts. For example, if the implementation of a method uses a local variable that uses the same name as a member variable of the class in which the method is declared, the local variable scope will take precedence and the class member variable will be effectively "hidden" unless qualified with the Self class instance reference:

```
interface

type

   TMyClass = class
      private
         MyVariable: String;
         procedure MyMethod;
      end;

implementation

procedure TMyClass.MyMethod;
var
   MyVariable: String; // This declaration overrides the scope of the class
begin
   MyVariable := 'Test';       // This will be assigned to the local
                               // MyVariable variable
   Self.MyVariable := 'Test';  // This will be assigned to the MyVariable
      member
                               // variable of the TMyClass class
end;
```

Certain naming conflicts are impossible because the compiler will not permit them. For example, you cannot give a local variable the same name as a parameter in the same function/procedure declaration, nor can you name a local variable the special "Result" variable name used for returning results from functions.

8.20 Casting Types

Casting is the process of converting a target value of one type to another type in order to use the value in a different type context. This is accomplished by enclosing the target value with the target type name and parentheses:

```
<Type Name>(<Target Value>)
```

There are type compatibility rules that determine whether a particular cast operation is valid, and invalid cast attempts will result in a compiler error. The following table details the various types and their valid target cast types:

Source Type	Valid Target Types
Integer	Integer Boolean Double DateTime Enumeration
Double	Double
String	String Char
Char	Char String
DateTime	DateTime Integer
Boolean	Boolean Integer
Enumeration	Enumeration Integer
Class Instance Class Type	Any same class type or ancestor class type

Casting is particularly useful for functions or procedures that accept a parameter of a base class type, but need to act on the various descendant class types in specific ways. For example, the following code shows how one would use the **is** operator to determine if the parameter is of the TComponent type and, if so, displays its name:

```
procedure DisplayName(Value: TObject);
begin
   if (Value is TComponent) then
      ShowMessage(TComponent(Value).Name);
end;
```

8.21 Exception Handling

Exceptions are special classes that represent an error that has been raised by code in the application, or by the execution environment/operating system itself. All exceptions descend from the ancestor Exception class, which itself descends from the TObject class. The declaration for the Exception class is as follows:

```
Exception = class(TObject)
   public
      property Message: String read;
      property UnitName: String read;
      property StackTrace: String read;
   end;
```

With client browser applications, all native JavaScript Error exceptions are automatically re-bound as Exception instances at runtime so that consistent exception class type interrogation/testing is possible with code that is shared between client and server applications. For server applications, all native execution environment exceptions are created using the Exception class, so no re-binding is necessary.

Raising Exceptions

You can raise an exception at any time by using the **raise** statement. The raise statement requires an Exception class (or descendant) instance as its only argument:

```
raise <Exception Class Instance>;
```

Once an exception is raised, execution stops immediately and the process of unwinding the call stack and triggering exception handlers begins.

The following example shows how to raise an exception that indicates that a parameter was not assigned a valid positive value:

```
function AddValues(A,B: Integer): Integer;
begin
   if (A < 0) then
      raise Exception.Create('First parameter '+IntToStr(A)+' cannot be
      negative');
   if (B < 0) then
      raise Exception.Create('Second parameter '+IntToStr(A)+' cannot be
      negative');
   Result:=(A+B);
end;
```

Handling Exceptions

Once an exception has been raised, execution immediately stops and the call stack is unwound, with any exception-handling blocks executed as necessary during this process.

Exceptions can be handled by using a **try..except** code block. The syntax of a try..except code block is:

```
try
   <Statements>
except
   <Exception-handling statements>
end;
```

A try..except code block catches any exceptions that occur with the try and except keywords, preventing them from escaping the current function or procedure and unwinding the call stack.

You can access the current exception from inside the except portion of the try..except code block by using the **on** statement, which uses the following syntax:

```
on <ExceptionInstanceVariable>: <ExceptionClass> do
   <Statements>

on <ExceptionClass> do
   <Statements>
```

There are two different variations of the on statement:

- The first variation specifies a local variable name followed by a colon and an exception class name. This is the most useful type of on statement because it allows you to capture the existing exception instance in a local variable. This is important when you want to examine the exception properties or log them for later examination. The exception class name is used to filter which exception classes are handled by the on statement.
- The second variation specifies an exception class name only and, again, the exception class name is used to filter which exception classes are handled by the on statement.

The following is an example of using the on statement to log an error message in the IDE using the LogOutput method:

```
begin
  try
     // Statements that raise exception
  except
     on E: Exception do
        LogOutput(E.Message);
  end;
end;
```

Please see the Debugging topic for more information on the LogOutput method.

Re-Raising Exceptions

You can re-raise an existing exception by using the **raise** statement without any arguments. Having the ability to re-raise exceptions is useful in situations where you want to do something with an exception, such as log its associated properties, before allowing the call stack to proceed unwinding.

> **Note**
> Re-raising exceptions can only be done from within the except portion of a try..except code block, and an attempt to do so outside of this context will cause a compiler error.

The following example expands upon the above example by also re-raising the exception after logging the error message:

```
begin
  try
     // Statements that raise exception
  except
     on E: Exception do
        LogOutput(E.Message);
     raise;
  end;
end;
```

Ensuring Code Execution After Exceptions

It is often necessary to ensure that certain statements execute, regardless of whether an exception is raised or not. This is accomplished by using a **try..finally** code block. The syntax of a try..finally code block is:

```
try
   <Statements>
finally
   <Statements>
end;
```

Any statements specified within the finally portion of the try..finally code block will always be executed, which is useful for situations where class instances, or other types of resources, have been allocated and need to be disposed of.

The following example shows the method of a class that toggles an internal Boolean variable in the class, and must ensure that the variable is toggled again before the method exits:

```
procedure TMyClass.Execute;
begin
   FExecuting:=True;
   try
      // Executing
   finally
      FExecuting:=False;
   end;
end;
```

Note
A try..finally code block also applies to the **exit** statement. If an exit statement is specified inside of a try..finally code block, the finally portion of the code block will be executed before the function or procedure actually exits.

Visual Client Application Exceptions

If an exception is not handled at run-time with a try..except code block in a visual application, the exception will

result in an message dialog appearing with the error message. If you do not want this to occur, you can define a TApplication OnError event handler for the global Application instance that is automatically created for visual client applications. Returning True from this event handler will indicate that the error was handled and will prevent the application from displaying an error dialog.

8.22 External Interfaces

Sometimes it is necessary to make calls from an application to code that is external to the application or component library code. In order to do so, you must first create an external interface that provides the compiler with information about how the external code should be used. An external interface is a set of declarations that differ slightly from normal declarations and contains no implementation code for the external declarations. Regular declarations and external declarations can be mixed together in the same source unit.

For client applications, external code will include JavaScript code or the built-in classes available as part of the a browser's DOM (Document Object Model) class hierarchy. The DOM is the core framework in a browser that allows any JavaScript code to create and manipulate elements in an HTML or XML document, as well as parts of the browser itself. A fairly complete external interface to the DOM is in the WebDOM unit that is included as part of the client component library.

Warning
Because of the nature of the JavaScript execution environment in browsers, any errors in external interface declarations will not be detected at compile time and will surface as run-time compiler or execution errors.

For server applications, external code is the native code surfaced as intrinsic functions/procedures in the run-time, as well as any external native Delphi code that is made available to the application code using native external module(s). Please see the Web Server Applications topic for more information on native external modules and the Creating a New External Module Interface Unit topic for more information on how to automatically generate the external interface for a native external module.

Warning
All external interface declarations are bound at compile time in server applications. This means that any native external modules must be available and loadable or the server application compilation will fail and the server application will not be able to handle any web server requests.

Please see the Using the Project Manager topic for more information on including external JavaScript source files with client applications and native external modules with server applications.

External Declarations

External declarations are marked as such by including the special **external** keyword before the declaration. The syntax for all declarations is:

```
{ Enumerated type declaration }

external <Type Name> = (<Member>[, Member]) [module <Native External Module>];


{ Class type declaration }

external <Class Name> [emit <External Type Name>]|[module <Native External
      Module>] = class[(<Class Name>)]
   [public]
      property <Property Name>:<Type Name> read [write]; [default;]
   end;

{ Function/procedure declarations }
```

```
external procedure <Function/Procedure Name>([<Parameter>[, <Parameter>]])[:
      <Type Name>] [module <Native External Module>];

{ Variable declarations }

var external <Variable Name>: <Type Name> [module <Native External Module>];
```

The following are the rules and restrictions for external declarations:

- Because JavaScript is the target language for external code in client applications and is case-sensitive, all external declarations for client applications are **case-sensitive** and must match the required case of the external JavaScript declarations for the same entities. This only applies to the external declarations, themselves. All application and component library code that **calls** the external code can still use any case desired, and the compiler will automatically make sure that the proper case is used in the emitted code for the application.
- Constant declarations are not discoverable in external JavaScript code or native external modules and are not considered part of an external interface. If there are any special constant declarations required in the external interface, they will need to be added manually.
- The ancestor class for all external class type declarations is the **TExternalObject** class, not the **TObject** class that is used with non-external class type declarations. If the ancestor class name is not explicitly specified, then the TExternalObject class is implicitly used as the ancestor class.
- External classes can only inherit from other external classes, and non-external classes can only inherit from non-external classes.
- You can use the **emit** clause with client external interfaces to control the class name/namespace used when the compiler emits references to the class when creating new instances. This is useful when instantiating JavaScript classes that are nested within namespaces. For example, Google Maps integration requires the following emit clause:

  ```
  external TGoogleMapOptions emit google.maps.MapOptions = class
  ```

- The **module** clause is required for external interfaces to native external modules in server applications. The clause tells the compiler which native external module contains the corresponding implementation for the declaration.
- External class type declarations contain only public member declarations. The public scope keyword is optional for external class type declarations and can be omitted.
- The read and write clauses for properties in external class type declarations do not refer to any member variables or methods.
- External class instances do not necessarily follow the same instantiation rules regarding the initialization of member variables. Non-external classes are guaranteed to have all of their member variables initialized to appropriate values, but this is not necessarily true for external classes.
- External classes can still be created with the Create method and freed with the Free method, but internally the compiler will emit slightly different code than it does for non-external classes.

The following is an example class type declaration from the WebDOM unit included with the component library for client applications:

```
external THTMLImageElement emit HTMLImageElement = class(THTMLElement)
```

```
   public
      { Properties }
      property alt: String read write;
      property height: Integer read;
      property isMap: Boolean read write;
      property longDesc: String read write;
      property name: String read write;
      property naturalHeight: Integer read;
      property naturalWidth: Integer read;
      property src: String read write;
      property useMap: String read write;
      property width: Integer read;
   end;
```

The following is an example class type declaration from the example extclasstype native external module included with the component library for server applications:

```
external TVehicle module extclasstype = class(TExternalObject)
   public
      { Properties }
      property VIN: String read write;
      property VehicleType: TVehicleType read write;
      property Make: String read write;
      property Model: String read write;
      property Year: Integer read write;
      { Methods }
      function EstimateValue(ACondition: TVehicleCondition): Integer;
   end;
```

8.23 Debugging

Server applications can be remotely debugged in the IDE, whereas client applications cannot. Please see the Running and Debugging a Project topic for more information on debugging server applications in the IDE.

There are currently two ways to debug client applications at runtime in a web browser: using a runtime function to log debug messages to the Messages panel in the IDE, or using the built-in debugging facilities in the web browser of your choosing. You can use the debug message option with the embedded web browser in the IDE, but not the debugging facilities.

LogOutput Procedure

For simple debugging needs, make sure to include the WebHTTP unit in the uses clause of the unit that you wish to debug. Finally, call the LogOutput procedure where necessary, passing any debug messages to the procedure as a single String parameter:

```
procedure LogOutput(const Msg: String;
                    const LogURL: String=DEFAULT_LOG_URL);
```

By default, the LogOutput procedure will send all debug output to the web server by using the following URL:

```
http://localhost/log
```

Any messages passed to the LogOutput method will automatically appear in the Messages panel in the IDE.

> **Note**
> This also applies to applications that are run in an external browser session. As long as the application is accessed via a **localhost** URL and is being loaded from the **Internal** web server running in the IDE, all debug output will get routed to the Messages panel in the IDE.

If you are running the application in an external browser session on a completely different machine or device, then be sure to include the second parameter to the LogOutput procedure. This parameter should include the IP address/host name of the machine running the IDE. For example, if you were running the application in a Chrome browser on an Android tablet, and the IDE was running on the same LAN at IP address 192.168.0.2, then you would use the following value for the second LogOutput parameter:

```
http://192.168.0.2/log
```

Web Browser Debuggers

For more complex debugging needs, make sure that the **Compress Output** option is **not** enabled for the client application project, build the project, and then run the application in an external web browser session with the web browser's debugger enabled. With Internet Explorer, Edge, FireFox, and Chrome, you can access the debugger by using the F12 key to open the developer tools panel while in the browser. The one major downside to this type of debugging is that you must debug the emitted JavaScript code, and not the Object Pascal code. Fortunately, though, the two are very similar in content and layout, and so the emitted JavaScript code is usually fairly easy to debug.

8.24 Asynchronous Calls

Elevate Web Builder supports asynchronous procedure/function calls in client browser applications using the special **async** keyword. Asynchronous calls allow the application code to queue a procedure/function call in the browser so that it is run as part of the message queue processing for the main UI thread in the browser. Asynchronous calls are available **only** at runtime, and will cause a component library compilation error if used in any design-time code.

How Asynchronous Calls Are Executed

Because asynchronous calls are added to the message queue for the main UI thread in the browser, they are executed in a first-in, first-out (FIFO) manner. This means that there may be a delay between when the asynchronous call is made and when the call is actually executed. Also, asynchronous calls are emitted by the compiler as JavaScript **closures**. Closures are functions that are dynamically created and capture the entire run-time scope of their parent execution context. Whenever a closure is actually executed, it will do so using the same scope that was present when the closure was created. Closures are ideal for asynchronous calls, because they need to capture the state of all variables and parameters so that they are available when the call is actually executed.

Executing an Asynchronous Call

To make an asynchronous procedure/function call, simply preface the call with the async keyword. For example,

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   async CreatePanel(0);
end;
```

will queue up a call to the CreatePanel procedure so that it will run in the next round of message processing in the browser. Because the compiler will emit a closure for this call, the value of any local variables or parameters will be properly captured, even if the parent method that is calling the function/procedure has finished executing.

Mixing Synchronous/Asynchronous Calls

Because the main UI thread in the browser is used for executing all code, any **synchronous** code will execute **before** any asynchronous calls that are queued in the message queue. This is important to understand because it determines how you should combine synchronous and asynchronous calls to achieve the desired outcome.

For example, suppose that you want to create a large number of panels in a container, and want to show a progress dialog while the panels are created. To do this, you would normally do something like this:

```
procedure TForm1.CreatePanels;
var
   I: Integer;
begin
   for I:=1 to 100 do
      TPanel.Create(Self);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
   ShowProgress('Creating panels...');
   CreatePanels;
   HideProgress;
end;
```

However, if you were to execute the above code in the browser, you will see that the panels are created, but the progress dialog will never show. This is because the UI updates for the ShowProgress call will not be executed until any other currently-executing code has completed. In this case, this is the CreatePanels and HideProgress calls, so the ShowProgress UI updates will get merged with the HideProgress UI updates, and the progress dialog will never get shown (or will be shown/hidden so fast that you won't see it).

The key to fixing this problem is to allow the UI to update incrementally while we create the panels, and we'll use asynchronous calls to do so:

```
procedure TForm1.CreatePanel(I: Integer);
begin
   TPanel.Create(Self);
   Inc(I);
   if (I < 100) then
      async CreatePanel(I)
   else
      async HideProgress;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
   ShowProgress('Creating panels...');
   async CreatePanel(0);
end;
```

We **don't** want to use an asynchronous call to ShowProgress because we want it to be executed immediately so that it is the first UI update to occur. However, we do want to queue each CreatePanel call and the HideProgress call because doing so will force them to execute in-order after any UI updates from the ShowProgress call, as well as allow the UI to update during each panel creation.

This page intentionally left blank

Chapter 9
Function and Procedure Reference

9.1 Abs

Unit: Internal

Available In: Client and Server Applications

```
function Abs(Value: Double): Double

function Abs(Value: Integer): Integer
```

The **Abs** function returns the absolute value of the input parameter. The return value is the same type as the input parameter.

Examples

```
X := Abs(-10);  // X is 10
X := Abs(100);  // X is 100
```

9.2 AddTrailingPathDelim

Unit: WebCore

Available In: Client and Server Applications

```
function AddTrailingPathDelim(const Value: String;
                              Delimiter: String=SLASH): String
```

The **AddTrailingPathDelim** function adds the specified path delimiter to the end of the string input parameter, if the path delimiter does not already exist at the end of the string. The return value is a string.

Examples

```
X := AddTrailingPathDelim('https://www.elevatesoft.com');  // X is
      'https://www.elevatesoft.com/'
```

9.3 ArcCos

Unit: Internal

Available In: Client and Server Applications

```
function ArcCos(Value: Double): Double

function ArcCos(Value: Integer): Double
```

The **ArcCos** function returns the arccosine, or inverse cosine, of the input parameter, which must be between -1 and 1. The return value is a Double value between 0 and Pi radians.

Examples

```
X := ArcCos(0.23290);  // X is 1.335737700525506
```

9.4 ArcSin

Unit: Internal

Available In: Client and Server Applications

```
function ArcSin(Value: Double): Double

function ArcSin(Value: Integer): Double
```

The **ArcSin** function returns the arcsine of the input parameter, which must be between -1 and 1. The return value is a Double value between -Pi/2 and Pi/2 radians.

Examples

```
X := ArcSin(0.23290);  // X is 0.23505862626939056
```

9.5 ArcTan

Unit: Internal

Available In: Client and Server Applications

```
function ArcTan(Value: Double): Double

function ArcTan(Value: Integer): Double
```

The **ArcTan** function returns the arc tangent of the input parameter. The return value is a Double value between -Pi/2 and Pi/2 radians.

Examples

```
X := ArcTan(0.23290);  // X is 0.22882093498523032
```

9.6 ArcTan2

Unit: Internal

Available In: Client and Server Applications

```
function ArcTan2(Y, X: Double): Double

function ArcTan2(Y, X: Integer): Double
```

The **ArcTan2** function returns a value between -Pi and Pi radians that specifies the counter-clockwise angle between the positive X axis and the point represented by the X and Y input parameters.

Examples

```
X := ArcTan2(100,3000);  // X is 0.033320995878247196
```

9.7 Assigned

Unit: Internal

Available In: Client and Server Applications

```
function Assigned(Value: TObject): Boolean

function Assigned(Value: TClass): Boolean

function Assigned(Value: function of object): Boolean

function Assigned(Value: procedure of object): Boolean

function Assigned(Value: function): Boolean

function Assigned(Value: procedure): Boolean

function Assigned(const Value: array of <Type>): Boolean
```

Examples

The **Assigned** function returns whether the input parameter is nil or has been assigned a value. The input parameter must be an object instance, method reference (function or procedure of object), string, or array variable. The return value is a Boolean value.

Examples

```
var
   MyObject: TObject=nil;
begin
   X := Assigned(MyObject);  // X is False
   MyObject := TObject.Create;
   X := Assigned(MyObject);  // X is True
end
```

9.8 Base64Decode

Unit: Internal

Available In: Client and Server Applications

```
function Base64Decode(const Value: String): String
```

The **Base64Decode** function decodes the base64-encoded string input parameter. The return value is a String value.

> **Note**
> This function will throw an exception if the string input parameter is not a valid base64-encoded string.

Examples

```
Y := Base64Encode('Hello World');
X := Base64Decode(Y);  // X is 'Hello World'
```

9.9 Base64Encode

Unit: Internal

Available In: Client and Server Applications

```
function Base64Encode(const Value: String): String
```

The **Base64Encode** function base64-encodes the string input parameter. The return value is a String value.

Examples

```
Y := Base64Encode('Hello World');
X := Base64Decode(Y);  // X is 'Hello World'
```

9.10 BoolToStr

Unit: WebCore

Available In: Client and Server Applications

```
function BoolToStr(Value: Boolean): String
```

The **BoolToStr** function returns 'True' if the input parameter is true, and 'False' if the input parameter is False. The return value is a String value.

Examples

```
X := BoolToStr(True);  // X is 'True'
```

9.11 Ceil

Unit: Internal

Available In: Client and Server Applications

```
function Ceil(Value: Double): Integer

function Ceil(Value: Integer): Integer
```

The **Ceil** function returns the closest integer that is greater than or equal to the value of the input parameter. The return value is an Integer.

Examples

```
X := Ceil(-10.4);  // X is -10
X := Ceil(15.98);  // X is 16
```

9.12 ChangeFileExt

Unit: WebCore

Available In: Client and Server Applications

```
function ChangeFileExt(const Value: String; const NewExt: String): String
```

The **ChangeFileExt** function changes the file extension in the string input parameter to the new file extension. The string input parameter can contain both a path and a file name or just a file name, but the file name should include a file extension. The specified file extension should include the period (.) file extension delimiter. The return value is a String value containing the original path and/or file name with the new file extension.

Examples

```
X := ChangeFileExt('https://www.elevatesoft.com/index.html','.js');  // X is
      'https://www.elevatesoft.com/index.js'
```

9.13 Chr

Unit: Internal

Available In: Client and Server Applications

```
function Chr(Value: Integer): Char
```

The **Chr** function returns the character representation of the Unicode code point input parameter. The return value is a Char.

Examples

```
X := Chr(220);  // X is 'Ü'
```

9.14 ClassByName

Unit: Internal

Available In: Client and Server Applications

```
function ClassByName(const Value: String): TClass
```

The **ClassByName** function returns the class type whose identifier matches the string input parameter, without case-sensitivity. This function depends upon the run-time type information in client and server applications.

For client applications, the class type declaration must have been compiled with the $RTTI symbol defined in order for the class type to be available to this function. By default, all client applications are compiled with the $RTTI symbol defined, but it is possible for code in the application to undefine the symbol using the $UNDEFINE compiler directive and exclude specific class types from being exposed in the run-time type information.

For server applications, run-time type information is always enabled for all code, and all class type declarations are always available to this function.

If the class type specified does not exist in the run-time type information, the return value will be nil.

Examples

```
X := ClassByName('TControl');  // X is TControl class type
```

9.15 CompareStr

Unit: Internal

Available In: Client and Server Applications

```
function CompareStr(const A, B: String): Integer
```

The **CompareStr** function compares the A string input parameter with the B string input parameter with case-sensitivity. The comparison is locale-insensitive. The return value is an Integer value of -1 if A is less than B, 0 if A is equal to B, and 1 if A is greater than B.

Examples

```
X := CompareStr('Absolute', 'Baseball');  // X is -1
```

9.16 CompareText

Unit: Internal

Available In: Client and Server Applications

```
function CompareText(const A, B: String): Integer
```

The **CompareText** function compares the A string input parameter with the B string input parameter, without case-sensitivity. The comparison is locale-insensitive. The return value is an Integer value of -1 if A is less than B, 0 if A is equal to B, and 1 if A is greater than B.

Examples

```
X := CompareText('Absolute', 'ABSOLUTE');  // X is 0
```

9.17 ComputeHash

Unit: WebCore

Available In: Server Applications

```
function ComputeHash(HashType: THashType;
                     const Value: String): String
```

The **ComputeHash** function computes a cryptographic hash for the string input parameter. The hash type parameter determines the type of computed hash that is returned, and can be one of the following values:

Hash Type	Description
htMD5	Specifies that the computed hash will be a 128-bit MD-5 (Message Digest) hash. **Warning** MD-5 hashes are cryptographic hashes, but are **not** secure and should only be used for purposes such as computing checksums for data.
htSHA1	Specifies that the computed hash will be a 160-bit SHA-1 (Secure Hash Algorithm) hash. **Warning** SHA-1 hashes are cryptographic hashes, but are **not** secure and should only be used for purposes such as computing checksums for data.
htSHA256	Specifies that the computed hash will be a 256-bit SHA-2 (Secure Hash Algorithm) hash.
htSHA512	Specifies that the computed hash will be a 512-bit SHA-2 (Secure Hash Algorithm) hash.

Note
The THashType enumerated type is declared in the **WebSrvr** unit.

The return value is a hexadecimal String value containing the computed hash.

Examples

```
X := ComputeHash(htSHA256,'Hello World');  // X is
      '979AB6536128EE79BCBB8D0386F29429D09B706B5CA77BFA5497968187E304C3'
```

9.18 Copy

Unit: Internal

Available In: Client and Server Applications

```
function Copy(const Value: String; Index: Integer;
              Count: Integer): String

function Copy(const Value: String; Index: Integer): String

function Copy(const Value: String): String

function Copy(const Value: array of <Type>; Index: Integer;
              Count: Integer): array of <Type>

function Copy(const Value: array of <Type>; Index: Integer): array of <Type>

function Copy(const Value: array of <Type>): array of <Type>
```

The **Copy** function returns a portion of the Value string or array input parameter. The optional Index input parameter specifies where to start the copy, and the optional Count input parameter specifies the length to copy. If the Count input parameter is not specified, then the copy will proceed until the end of the Value input parameter. If neither the Index or Count input parameters are specified, then an exact copy of the Value input parameter will be returned.

> **Note**
> Please remember that indexes into String values are 1-based, whereas indexes into arrays are 0-based. For more information on these types, please see the Types topic.

Examples

```
X := Copy('abcdef', 4, 3);  // X is 'def'
X := Copy('abcdef', 2);  // X is 'bcdef'
X := Copy([10,20,30,40], 0, 3);  // X is [10,20,30]
```

9.19 Cos

Unit: Internal

Available In: Client and Server Applications

```
function Cos(Value: Double): Double

function Cos(Value: Integer): Double
```

The **Cos** function returns the cosine of the input parameter, which is an angle specified in radians. To convert an angle from degrees to radians, use the Radians function. The return value is a Double value between -1 and 1.

Examples

```
X := Cos(0.23290);  // X is 0.9730011668494914
```

9.20 CreateActiveXObject

Unit: Internal

Available In: Client Applications

```
function CreateActiveXObject(const ObjectClass: String): <External Object
      Instance>
```

The **CreateActiveXObject** function creates an external ActiveX object instance (Internet Explorer only) and returns it as the result.

> **Note**
> You must always cast the result of this function to the desired external class in order to be able to use the resulting instance in your code.

Examples

```
var
   TempDocument: TDocument;
begin
   // Must always cast the result to the desired class
   TempDocument:=TDocument(CreateActiveXObject('Microsoft.XMLDOM'));
   TempDocument.LoadXML(Value);
end
```

9.21 CreateDirectory

Unit: Internal

Available In: Server Applications

```
procedure CreateDirectory(const Directory: String)
```

The **CreateDirectory** procedure creates the directory specified in the input parameter. The specified directory path can contain intermediate directories, but those intermediate directories must exist and the procedure will only create the final directory in the path. If the directory cannot be created for any reason, an exception will be raised.

> **Note**
> The specified directory path should be an absolute path. Relative paths are permitted, but they rely on the current working directory. The current working directory is a per-process setting and is not reliable in a multi-threaded setting like the web server.

Examples

```
if (not DirectoryExists('C:\temp\backup')) then
   CreateDirectory('C:\temp\backup');
```

9.22 CreateObject

Unit: Internal

Available In: Client Applications

```
function CreateObject(const ObjectLiteral: String): <External Object
      Instance>
```

The **CreateObject** function creates an external object instance from a JavaScript object literal and returns it as the result. This function can be especially useful with JS APIs that require that you create object instances using object literals.

Note
You must always cast the result of this function to the desired external class in order to be able to reference any properties or methods of the new external class instance from within Elevate Web Builder code.

Warning
This function internaly uses the JavaScript eval function in order to create the object. Be very careful about passing object literal strings that have been derived from an external source to this function.

Examples

```
type
   external TMyExternalObject emit MyJSAPI.MyExternalObject = class
      public
         property Name: String read write;
      end;

var
   TempObject: TMyExternalObject;
begin
   TempObject := TMyExternalObject(CreateObject('{ Name: ''My External
      Object'' }'));
end
```

9.23 Date

Unit: Internal

Available In: Client and Server Applications

```
function Date: DateTime
```

The **Date** function returns the current date. The return value is a DateTime value.

> **Note**
> DateTime values encompass both the date and time portion for a given date/time. The Date function will set the time portion of the date/time to the beginning (exact midnight of the current/prior day) of the returned date.

Examples

```
X := DateToStr(Date);  // X is '2/13/2012'
```

9.24 DateTimeToStr

Unit: WebCore

Available In: Client and Server Applications

```
function DateTimeToStr(Value: DateTime; UTC: Boolean=False): String
```

The **DateTimeToStr** function returns a formatted local or UTC date and time string for the DateTime input parameter. The format of the string is determined by the TFormatSettings ShortDateFormat and ShortTimeFormat properties. The return value is a String value.

Examples

```
A := StrToDateTime('2/13/2012 12:10 PM');
X := DateTimeToStr(A);  // X is '2/13/2012 12:10 PM'
```

9.25 DateTimeToISOStr

Unit: Internal

Available In: Client and Server Applications

```
function DateTimeToISOStr(Value: DateTime): String
```

The **DateTimeToISOStr** function returns an ISO-8601-formatted date-time string value for the date-time input parameter. The return value is a String value.

Examples

```
X := DateTimeToISOStr(StrToDateTime('8/15/2015 12:40 PM'));  // X is
      '2015-08-15T16:40:00.000Z'
```

9.26 DateToStr

Unit: WebCore

Available In: Client and Server Applications

```
function DateToStr(Value: DateTime; UTC: Boolean=False): String;
```

The **DateToStr** function returns a formatted local or UTC date string for the DateTime input parameter. The format of the string is determined by the TFormatSettings ShortDateFormat property. The return value is a String value.

Examples

```
A := StrToDateTime('2/13/2012');
X := DateToStr(A);  // X is '2/13/2012'
```

9.27 DayOf

Unit: Internal

Available In: Client and Server Applications

```
function DayOf(Value: DateTime; UTC: Boolean=False): Integer
```

The **DayOf** function returns the day number of the input parameter in local or UTC time. The return value is an Integer value.

Examples

```
X := DayOf(Date);  // X is 13 (assuming a date of 02/13/2012)
```

9.28 Dec

Unit: Internal

Available In: Client and Server Applications

```
procedure Dec(var Value: Integer)

procedure Dec(var Value: Integer; By: Integer)
```

The **Dec** procedure decrements the Integer input parameter by 1, or by the By input parameter, if specified.

Examples

```
X := 10;
Dec(X);
Y := IntToStr(X)  // Y is '9'
```

9.29 DecodeURL

Unit: WebCore

Available In: Client and Server Applications

```
function DecodeURL(const URL: String): String
```

The **DecodeURL** function returns the decoded version of the encoded URL input parameter. The return value is a String value.

This function is the analog of the EncodeURL function.

Examples

```
X := DecodeURL('https://localhost/home%20page.html');  // X is
      'https://localhost/home page.html'
```

9.30 DecodeURLComponent

Unit: WebCore

Available In: Client and Server Applications

```
function DecodeURLComponent(const Component: String): String
```

The **DecodeURLComponent** function returns the decoded version of the encoded URL component input parameter. A URL component is any portion of a URL within a path delimiter (/), query delimiter (?), or parameter delimiter (&). The return value is a String value.

This function is the analog of the EncodeURLComponent function.

Examples

```
X := DecodeURLComponent('Jones%20%26%20Smith');  // X is 'Jones & Smith'
```

9.31 DecryptStr

Unit: WebCore

Available In: Server Applications

```
function DecryptStr(const Value: String; const Password: String;
                    EncryptionType: TEncryptionType=etAES128): String;
```

The **DecryptStr** function decrypts an encrypted string (in hexadecimal format) using the specified encryption algorithm.

> **Note**
> The TEncryptionType enumerated type is declared in the **WebSrvr** unit.

The return value is a String value containing the decrypted string.

Examples

```
X := DecryptStr('A98649088318CB72B224B222FC10BC26','Test Password',etAES256);
       // X is 'Hello World'
```

9.32 Defined

Unit: Internal

Available In: Client Applications

```
function Defined(<Object Property>): Boolean
```

Examples

The **Defined** function returns whether the object property input parameter is undefined, and is used with external JavaScript objects to determine if a given property exists in a JavaScript object. The input parameter must be a property reference for a valid object instance. The return value is a Boolean value.

Examples

```
type

   external TExternalText = class(TExternalObject)
      public
         property Text: String read write;
         property UseCount: Integer read write;
      end;

var
   MyObject: TExternalText;
begin
   MyObject := TExternalText(CreateObject('{ Text: "Hello World" }'));
   X := Defined(MyObject.Text);  // X is True
   X := Defined(MyObject.UseCount);  // X is False
end
```

9.33 Degrees

Unit: Internal

Available In: Client and Server Applications

```
function Degrees(Value: Double): Double

function Degrees(Value: Integer): Double
```

The **Degrees** function converts the input parameter, which is an angle specified in radians, to degrees. The return value is a Double value.

Examples

```
X := Degrees(92.398);  // X is 5294.01543544978
```

9.34 Delete

Unit: Internal

Available In: Client and Server Applications

```
procedure Delete(const Value: array of <Type>; Index: Integer;
                 Count: Integer)

procedure Delete(const Value: array of <Type>; Index: Integer)
```

The **Delete** procedure deletes a portion of the Value array input parameter. The optional Index input parameter specifies where to start the deletion, and the optional Count input parameter specifies the number of array elements to delete. If the Count input parameter is not specified, then the deletion will proceed until the end of the Value input parameter.

Note
This procedure cannot be used with strings in Elevate Web Builder. Strings are immutable in JavaScript, and therefore cannot be modified in-place using procedures such as this. For more information on these types, please see the Types topic.

Examples

```
X := [10,20,30,40];
Delete(X, 2, 2); // X is [10,20] after the Delete call
```

9.35 DirectoryExists

Unit: Internal

Available In: Server Applications

```
function DirectoryExists(const Directory: String): Boolean
```

The **DirectoryExists** function returns True if the directory specified in the input parameter exists, or False if the directory does not exist.

> **Note**
> The specified directory path should be an absolute path. Relative paths are permitted, but they rely on the current working directory. The current working directory is a per-process setting and is not reliable in a multi-threaded setting like the web server.

Examples

```
if DirectoryExists('C:\temp\backup') then
   RemoveDirectory('C:\temp\backup');
```

9.36 DoubleToStr

Unit: Internal

Available In: Client and Server Applications

```
function DoubleToStr(Value: Double; Decimals: Integer=-1): String
```

The **DoubleToStr** function returns a formatted string for the Double input parameter. The decimal separator used in the formatted string is **always** a period (.). The return value is a String value. You can use the optional Decimals parameter to specify that the return value is formatted to a specific number of decimal places.

Examples

```
A := StrToDouble('1200.548');
X := DoubleToStr(A);  // X is '1200.548'

A := StrToDouble('1200.548');
X := DoubleToStr(A, 1);  // X is '1200.5'
```

9.37 DST

Unit: Internal

Available In: Client and Server Applications

```
function DST(Value: DateTime; UTC: Boolean=False): Boolean
```

The **DST** function returns True if the the date/time input parameter falls into a period of daylight-savings time, and False if it doesn't.

Examples

```
X := DST(Now);  // X is False for '12/20/2020 1:23 PM'
```

9.38 EncodeDate

Unit: Internal

Available In: Client and Server Applications

```
function EncodeDate(Year: Integer; Month: Integer; Day: Integer;
                    UTC: Boolean=False): Integer
```

The **EncodeDate** function returns the local or UTC date from the year, month, and day input parameters. The return value is a DateTime value.

Examples

```
X := DateToStr(EncodeDate(2012,2,13));  // X is '2/13/2012'
```

9.39 EncodeDateTime

Unit: Internal

Available In: Client and Server Applications

```
function EncodeDateTime(Year: Integer; Month: Integer; Day: Integer;
                        Hour: Integer; Minute: Integer; Second: Integer;
                        MSecond: Integer; UTC: Boolean=False): Integer
```

The **EncodeDateTime** function returns the local or UTC date and time from the year, month, day, hour, minute, second, and millisecond input parameters. The return value is a DateTime value.

Examples

```
X := DateTimeToStr(EncodeDateTime(2012,2,13,12,10,0,0)); // X is '2/13/2012
                                                          // 12:10 PM'
```

9.40 EncodeTime

Unit: Internal

Available In: Client and Server Applications

```
function EncodeTime(Hour: Integer; Minute: Integer; Second: Integer; MSecond:
      Integer;
                    UTC: Boolean=False): Integer
```

The **EncodeTime** function returns the local or UTC time from the hour, minute, second, and millisecond input parameters. The return value is a DateTime value.

Examples

```
X := TimeToStr(EncodeTime(12,10,0,0));  // X is '12:10 PM'
```

9.41 EncodeURL

Unit: WebCore

Available In: Client and Server Applications

```
function EncodeURL(const URL: String): String
```

The **EncodeURL** function returns the encoded version of the URL input parameter. All characters are encoded **except** for the following:

```
A-Z a-z 0-9 ; , / ? : @ & = + $ - _ . ! ~ * ' ( ) #
```

The return value is a UTF-8-encoded String value containing percent-encoded hex strings for any encoded characters.

This function is the analog of the DecodeURL function.

> **Note**
> This function is not sufficient for encoding parameter values in a URL because the ampersand (&), plus (+), and equals (=) characters are not encoded. Please see the EncodeURLComponent function for more information on encoding parameter values for inclusion in URLs.

Examples

```
X := EncodeURL('https://localhost/home page.html');  // X is
      'https://localhost/home%20page.html'
```

9.42 EncodeURLComponent

Unit: WebCore

Available In: Client and Server Applications

```
function EncodeURLComponent(const Component: String): String
```

The **EncodeURLComponent** function returns the encoded version of the URL component input parameter. A URL component is any portion of a URL within a path delimiter (/), query delimiter (?), or parameter delimiter (&). All characters are encoded **except** for the following:

```
A-Z a-z 0-9 - _ . ! ~ * ' ( )
```

The return value is a UTF-8-encoded String value containing percent-encoded hex strings for any encoded characters.

This function is the analog of the DecodeURLComponent function.

Examples

```
X := EncodeURLComponent('Jones & Smith');  // X is 'Jones%20%26%20Smith'
```

9.43 EncryptStr

Unit: WebCore

Available In: Server Applications

```
function EncryptStr(const Value: String; const Password: String;
                    EncryptionType: TEncryptionType=etAES128): String;
```

The **EncryptStr** function encrypts a string using the specified encryption algorithm.

> **Note**
> The TEncryptionType enumerated type is declared in the **WebSrvr** unit.

The return value is a hexadecimal String value containing the encrypted string.

Examples

```
X := DecryptStr('Hello World','Test Password',etAES256);  // X is
      'A98649088318CB72B224B222FC10BC26'
```

9.44 EnsureFileExt

Unit: WebCore

Available In: Client and Server Applications

```
function EnsureFileExt(const Value: String; const OldExt: String;
                       const NewExt: String): String
```

The **EnsureFileExt** function is very similar to the ChangeFileExt function. However, instead of just changing the file extension to the new file extension, the function first checks the existing file extension to ensure that it matches the old file extension:

- If the existing file extension matches the old file extension, then the file extension is changed to the new file extension.
- If the existing file extension does not match the old file extension, then the new file extension is appended to the string input parameter.

The string input parameter can contain both a path and a file name or just a file name, but the file name should include a file extension. The specified file extension should include the period (.) file extension delimiter. The return value will be a string containing the original path and/or file name with the new file extension.

Examples

```
X := EnsureFileExt('https://www.elevatesoft.com/index.html','.html','.js');
      // X is 'https://www.elevatesoft.com/index.js'
```

9.45 EscapeJSON

Unit: WebCore

Available In: Client and Server Applications

```
function EscapeJSON(const Value: String): String
```

The **EscapeJSON** function returns the escaped version of the JSON input parameter so that the JSON can be included as part of a JSON string property value. The following characters are escaped by prefixing the characters with the backslash (\) character:

```
" \ /
```

In addition, any non-printable characters whose ordinal value is less than the space character (' ') are escaped using the following format:

```
"\u<4-Digit Hex String>
```

The return value is a String value.

This function is the analog of the UnEscapeJSON function.

Examples

```
X := EscapeJSON('"Hello World"');  // X is '\"Hello World\"'
```

9.46 Exp

Unit: Internal

Available In: Client and Server Applications

```
function Exp(Value: Double): Double

function Exp(Value: Integer): Double
```

The **Exp** function returns e raised to the power specified by the input parameter, where e is the base of the natural logarithm. The return value is a Double value.

Examples

```
X := Exp(0.523);  // X is 1.6870813093472114
```

9.47 ExtractFileExt

Unit: WebCore

Available In: Client and Server Applications

```
function ExtractFileExt(const Value: String): String
```

The **ExtractFileExt** function returns the file extension, including the period (.) file extension delimiter, in the string input parameter. The return value will be a string containing the file extension, if one exists.

Examples

```
X := ExtractFileExt('https://www.elevatesoft.com/index.html');  // X is
      '.html'
```

9.48 ExtractFileName

Unit: WebCore

Available In: Client and Server Applications

```
function ExtractFileName(const Value: String;
                         Delimiter: String=SLASH): String
```

The **ExtractFileName** function returns the file name, including the file extension, in the string input parameter. The return value will be a string containing the file name, if one exists.

Examples

```
X := ExtractFileName('https://www.elevatesoft.com/index.html');  // X is
      'index.html'
```

9.49 ExtractFileRoot

Unit: WebCore

Available In: Client and Server Applications

```
function ExtractFileRoot(const Value: String; const Ext: String;
                         Delimiter: String=SLASH): String
```

The **ExtractFileRoot** function returns the file name, without the file extension, in the string input parameter. The specified file extension should include the period (.) file extension delimiter. The return value will be a string containing the root of the file name, if one exists.

Examples

```
X := ExtractFileRoot('https://www.elevatesoft.com/index.html,'.html');  // X
      is 'index'
```

9.50 ExtractPath

Unit: WebCore

Available In: Client and Server Applications

```
function ExtractPath(const Value: String;
                     Delimiter: String=SLASH): String
```

The **ExtractPath** function returns the path in the string input parameter. The return value will be a string containing the path, if one exists.

Examples

```
X := ExtractPath('https://www.elevatesoft.com/index.html');  // X is
      'https://www.elevatesoft.com'
```

9.51 FileAttr

Unit: Internal

Available In: Server Applications

```
function FileAttr(const FileName: String): Integer
```

The **FileAttr** function returns the set of all attributes of the file input parameter. The file attributes are operating system-specific, so please refer to the documentation for the operating system in use with the web server for more information on how to interpret the returned file attributes. The return value is an Integer value.

> **Note**
> The specified file name should contain an absolute path. Relative paths are permitted, but they rely on the current working directory. The current working directory is a per-process setting and is not reliable in a multi-threaded setting like the web server.

Examples

```
X := FileAttr('C:\temp\test.html');  // X is 32
```

9.52 FileCreation

Unit: Internal

Available In: Server Applications

```
function FileCreation(const FileName: String): DateTime
```

The **FileCreation** function returns the creation date and time of the file input parameter. The return value is a DateTime value.

> **Note**
> The specified file name should contain an absolute path. Relative paths are permitted, but they rely on the current working directory. The current working directory is a per-process setting and is not reliable in a multi-threaded setting like the web server.

Examples

```
X := FileCreation('C:\temp\test.html');  // X is 1604515489696, or '11/4/2020
      1:44 PM'
```

9.53 FileDelete

Unit: Internal

Available In: Server Applications

```
procedure FileDelete(const FileName: String)
```

The **FileDelete** procedure deletes the file specified in the input parameter. If the file cannot be deleted for any reason, an exception will be raised.

> **Note**
> The specified file name should contain an absolute path. Relative paths are permitted, but they rely on the current working directory. The current working directory is a per-process setting and is not reliable in a multi-threaded setting like the web server.

Examples

```
if FileExists('C:\temp\test.html') then
   FileDelete('C:\temp\test.html');
```

9.54 FileExists

Unit: Internal

Available In: Server Applications

```
function FileExists(const FileName: String): Boolean
```

The **FileExists** function returns True if the file input parameter exists, and False if the file does not exist.

> **Note**
> The specified file name should contain an absolute path. Relative paths are permitted, but they rely on the current working directory. The current working directory is a per-process setting and is not reliable in a multi-threaded setting like the web server.

Examples

```
X := FileExists('C:\temp\test.html');  // X is True
```

9.55 FileModification

Unit: Internal

Available In: Server Applications

```
function FileModification(const FileName: String): DateTime
```

The **FileModification** function returns the last modification date and time of the file input parameter. The return value is a DateTime value.

> **Note**
> The specified file name should contain an absolute path. Relative paths are permitted, but they rely on the current working directory. The current working directory is a per-process setting and is not reliable in a multi-threaded setting like the web server.

Examples

```
X := FileModification('C:\temp\test.html');  // X is 1604515489696, or
      '11/4/2020 1:44 PM'
```

9.56 FileRename

Unit: Internal

Available In: Server Applications

```
procedure FileRename(const FileName: String;
                     const NewFileName: String)
```

The **FileRename** procedure renames the file specified in the input parameter to the specified new file name. If the file cannot be renamed for any reason, an exception will be raised.

> **Note**
> The specified file names should contain absolute paths. Relative paths are permitted, but they rely on the current working directory. The current working directory is a per-process setting and is not reliable in a multi-threaded setting like the web server.

Examples

```
if FileExists('C:\temp\test.html') and (not
      FileExists('C:\temp\newtest.html')) then
   FileRename('C:\temp\test.html','C:\temp\newtest.html');
```

9.57 FileSize

Unit: Internal

Available In: Server Applications

```
function FileSize(const FileName: String): Integer
```

The **FileSize** function returns the size, in bytes, of the file input parameter. The return value is an Integer value.

> **Note**
> The specified file name should contain an absolute path. Relative paths are permitted, but they rely on the current working directory. The current working directory is a per-process setting and is not reliable in a multi-threaded setting like the web server.

Examples

```
X := FileSize('C:\temp\test.html');  // X is 25547
```

9.58 FloatToStr

Unit: WebCore

Available In: Client and Server Applications

```
function FloatToStr(Value: Double; Decimals: Integer=-1): String
```

The **FloatToStr** function returns a formatted string for the Double input parameter. The decimal separator used in the formatted string is determined by the TFormatSettings DecimalSeparator property. The optional Decimals input parameter determines the number of decimal places used in the formatted string. The return value is a String value.

Examples

```
A := StrToFloat('1200.548');
X := FloatToStr(A);  // X is '1200.548'

A := StrToFloat('1200.548');
X := FloatToStr(A,2);  // X is '1200.55'
```

9.59 Floor

Unit: Internal

Available In: Client and Server Applications

```
function Floor(Value: Double): Integer

function Floor(Value: Integer): Integer
```

The **Floor** function returns the closest integer that is less than or equal to the value of the input parameter. The return value is an Integer.

Examples

```
X := Floor(-10.4);  // X is -11
X := Floor(15.98);  // X is 15
```

9.60 GenerateRandom

Unit: WebCore

Available In: Server Applications

```
function GenerateRandom(Length: Integer): String
```

The **GenerateRandom** function generates a cryptographically-secure random number with the specified length. The return value will be a hexadecimal string containing the generated random number.

Examples

```
X := GenerateRandom(32);  // X is
      '6F1F259CC46DB8D76B900C3ED88772402E74E2FC82E392CEC1D6A5CCDE824F07'
```

9.61 HideProgress

Unit: WebForms

Available In: Client Applications

```
procedure HideProgress
```

The **HideProgress** procedure decrements the global progress reference count, and if the reference count is 0, hides the active progress dialog. The ShowProgress procedure shows a progress dialog and increments the progress reference count.

Examples

```
HideProgress;
```

9.62 HourOf

Unit: Internal

Available In: Client and Server Applications

```
function HourOf(Value: DateTime; UTC: Boolean=False): Integer
```

The **HourOf** function returns the hour number of the input parameter in local or UTC time. The return value is an Integer value between 0 (midnight) and 23 (11:00 PM).

Examples

```
X := HourOf(Time);  // X is 12 (assuming a time of 12:10 PM)
```

9.63 Inc

Unit: Internal

Available In: Client and Server Applications

```
procedure Inc(var Value: Integer)

procedure Inc(var Value: Integer; By: Integer)
```

The **Inc** procedure increments the Integer input parameter by 1, or by the By input parameter, if specified.

Examples

```
X := 1;
Inc(X);
Y := IntToStr(X)  // Y is '2'
```

9.64 InheritsFrom

Unit: Internal

Available In: Client and Server Applications

```
function InheritsFrom(AChildClass: TClass; AClass: TClass): Boolean;

function InheritsFrom(AObject: TObject; AClass: TClass): Boolean;
```

The **InheritsFrom** function returns True if the class or class instance input parameter is a descendant class of the specified class, or False if not.

Examples

```
var
   MyEdit: TEdit;
begin
   X := InheritsFrom(TServerRequest,TControl);  // X is False
   MyEdit := TEdit.Create(MyForm);
   X := InheritsFrom(MyEdit,TControl);  // X is True
end
```

9.65 Insert

Unit: Internal

Available In: Client and Server Applications

```
procedure Insert(Value: <Type>; const Array: array of <Type>;
                 Index: Integer)

procedure Insert(const Value: array of <Type>; const Array: array of <Type>;
                 Index: Integer)
```

The **Insert** procedure inserts a new value (or array of values) into the Array input parameter. The Index input parameter specifies where the insertion will take place. The Value input parameter must be type-compatible with the Array input parameter that it is being inserted into.

> **Note**
> This procedure cannot be used with strings in Elevate Web Builder. Strings are immutable in JavaScript, and therefore cannot be modified in-place using procedures such as this. For more information on these types, please see the Types topic.

Examples

```
X := [10,20,30,40];
Insert(35, X, 3); // X is [10,20,30,35,40] after the Insert call

X := [10,20,30,40];
Y := [21,22,23,24,25];
Insert(Y, X, 2); // X is [10,20,21,22,23,24,25,30,40] after the Insert call
```

9.66 IntToHex

Unit: Internal

Available In: Client and Server Applications

```
function IntToHex(Value: Integer; Digits: Integer): String
```

The **IntToHex** function returns a formatted hexadecimal string for the Integer input parameter. The Digits input parameter indicates the minimum length of the String return value.

Examples

```
X := IntToHex(1052);  // X is '041C'
```

9.67 IntToStr

Unit: Internal

Available In: Client and Server Applications

```
function IntToStr(Value: Integer): String
```

The **IntegerToStr** function returns a formatted string for the Integer input parameter. The return value is a String value.

Examples

```
X := IntToStr(-102);  // X is '-102'
```

9.68 IsAlpha

Unit: WebCore

Available In: Client and Server Applications

```
function IsAlpha(const Value: Char): Boolean
```

The **IsAlpha** function returns True if the input parameter is an alpha-numeric character:

```
A..Z a..z
```

Examples

```
X := IsAlpha('A');  // X is True
```

9.69 IsBool

Unit: WebCore

Available In: Client and Server Applications

```
function IsBool(const Value: String): Boolean
```

The **IsBool** function returns True if the input parameter is a valid boolean string:

```
'True'
'False'
```

Examples

```
X := IsBool('Hello');  // X is False
```

9.70 IsDate

Unit: WebCore

Available In: Client and Server Applications

```
function IsDate(const Value: String): Boolean
```

The **IsDate** function returns True if the input parameter is a valid date string. The required format of the string is determined by the TFormatSettings ShortDateFormat property.

Examples

```
X := IsDate('02/13/2012');  // X is True
```

9.71 IsDateTime

Unit: WebCore

Available In: Client and Server Applications

```
function IsDateTime(const Value: String): Boolean
```

The **IsDateTime** function returns True if the input parameter is a valid date/time string. The required format of the string is determined by the TFormatSettings ShortDateFormat and ShortTimeFormat properties.

Examples

```
X := IsDateTime('2/13/2012 12:10 PM');  // X is True
```

9.72 IsDay

Unit: WebCore

Available In: Client and Server Applications

```
function IsDay(Month: Integer; Value: Integer; LeapYear: Boolean=False):
      Boolean
```

The **IsDay** function returns True if the input parameter is a valid day number for the specified month and leap year parameters.

Examples

```
X := IsDay(3,31);  // X is True
```

9.73 IsDigit

Unit: WebCore

Available In: Client and Server Applications

```
function IsDigit(const Value: Char): Boolean
```

The **IsDigit** function returns True if the input parameter is a digit character:

```
0..9
```

Examples

```
X := IsDigit('9');  // X is True
```

9.74 IsFloat

Unit: WebCore

Available In: Client and Server Applications

```
function IsFloat(const Value: String): Boolean
```

The **IsFloat** function returns True if the input parameter is a valid floating-point string. The decimal separator used in the formatted string is determined by the TFormatSettings DecimalSeparator property.

Examples

```
X := IsFloat('200.10');  // X is True
```

9.75 IsHour

Unit: WebCore

Available In: Client and Server Applications

```
function IsHour(Value: Integer; AMPM: Boolean=False): Boolean
```

The **IsHour** function returns True if the input parameter is a valid hour number for the specified clock. If the AMPM parameter is false, the valid hour numbers are 0-23, and if the AMPM parameter is True, the valid hour numbers are 0-12.

Examples

```
X := IsHour(12);  // X is True
```

9.76 IsInt

Unit: WebCore

Available In: Client and Server Applications

```
function IsInt(const Value: String): Boolean
```

The **IsInt** function returns True if the input parameter is a valid integer string.

Examples

```
X := IsInt('10.20');  // X is False
```

9.77 IsLeapYear

Unit: WebCore

Available In: Client and Server Applications

```
function IsLeapYear(Value: Integer): Boolean
```

The **IsLeapYear** function returns True if the year input parameter is a leap year.

Examples

```
X := IsLeapYear(2020);  // X is True
```

9.78 IsMinute

Unit: WebCore

Available In: Client and Server Applications

```
function IsMinute(Value: Integer): Boolean
```

The **IsMinute** function returns True if the input parameter is a valid minute number. The valid minute numbers are 0-59.

Examples

```
X := IsMinute(62);  // X is False
```

9.79 IsMonth

Unit: WebCore

Available In: Client and Server Applications

```
function IsMonth(Value: Integer): Boolean
```

The **IsMonth** function returns True if the input parameter is a valid month number. The valid month numbers are 1-12.

Examples

```
X := IsMonth(0);  // X is False
```

9.80 IsMSecond

Unit: WebCore

Available In: Client and Server Applications

```
function IsMSecond(Value: Integer): Boolean
```

The **IsMSecond** function returns True if the input parameter is a valid millisecond number. The valid millisecond numbers are 0-999.

Examples

```
X := IsMSecond(317);  // X is True
```

9.81 IsSecond

Unit: WebCore

Available In: Client and Server Applications

```
function IsSecond(Value: Integer): Boolean
```

The **IsSecond** function returns True if the input parameter is a valid second number. The valid second numbers are 0-59.

Examples

```
X := IsSecond(34);  // X is True
```

9.82 IsTime

Unit: WebCore

Available In: Client and Server Applications

```
function IsTime(const Value: String): Boolean
```

The **IsTime** function returns True if the input parameter is a valid time string. The required format of the string is determined by the TFormatSettings ShortTimeFormat property.

Examples

```
X := IsTime('12:10 PM');  // X is True
```

9.83 ISOStrToDateTime

Unit: Internal

Available In: Client and Server Applications

```
function ISOStrToDateTime(const Value: String): DateTime
```

The **ISOStrToDateTime** function returns a date-time value for the ISO-8601-formatted date-time string input parameter. The return value is a DateTime value.

Examples

```
X := DateTimeToStr(ISOStrToDateTime('2015-08-15T16:40:31.601Z'));  // X is
      '8/15/2015 12:40 PM'
```

9.84 Join

Unit: Internal

Available In: Client and Server Applications

```
function Join(const Array: array of String; const Separator: String): String

function Join(const Array: array of String): String
```

The **Join** function builds a new string from the elements in the Array string array input parameter. The Separator input parameter is optional. If the Separator input parameter is specified, then the return value is a String value that contains all string elements from the array, separated by the Separator input parameter. If the Separator input parameter is not specified, then the return value is a String value that contains all string elements from the array.

Examples

```
X := Join(['Hello,','my','name','is','Jim'], ' ');  // X is 'Hello, my name
      is Jim'
```

9.85 Length

Unit: Internal

Available In: Client and Server Applications

```
function Length(const Value: String): Integer

function Length(const Value: array of <Type>): Integer
```

The **Length** function returns the length of the String or array input parameter. The return value is an Integer value.

Examples

```
X := Length('How long is this string');  // X is 23
```

9.86 Ln

Unit: Internal

Available In: Client and Server Applications

```
function Ln(Value: Double): Double

function Ln(Value: Integer): Double
```

The **Ln** function returns the natural logarithm of the input parameter, which must be greater than 0. The return value is a Double value.

Examples

```
X := Ln(0.523);  // X is -0.6481738149172141
```

9.87 LocaleCompareStr

Unit: Internal

Available In: Client and Server Applications

```
function LocaleCompareStr(const A, B: String): Integer
```

The **LocaleCompareStr** function compares the A string input parameter with the B string input parameter with case-sensitivity. The comparison uses the browser's current locale setting to compare the two strings. The return value is an Integer value of -1 if A is less than B, 0 if A is equal to B, and 1 if A is greater than B.

Examples

```
X := LocaleCompareStr('Absolute', 'Baseball');  // X is -1
```

9.88 LocaleCompareText

Unit: Internal

Available In: Client and Server Applications

```
function LocaleCompareText(const A, B: String): Integer
```

The **LocaleCompareText** function compares the A string input parameter with the B string input parameter without case-sensitivity. The comparison uses the browser's current locale setting to compare the two strings. The return value is an Integer value of -1 if A is less than B, 0 if A is equal to B, and 1 if A is greater than B.

Examples

```
X := LocaleCompareText('Absolute', 'ABSOLUTE');  // X is 0
```

9.89 LocaleLowerCase

Unit: Internal

Available In: Client and Server Applications

```
function LocaleLowerCase(const Value: String): String
```

The **LocaleLowerCase** function returns the Value input parameter with all characters converted to their lower-case representation. The browser's current locale setting is used to perform this conversion. The return value is a String value.

Examples

```
X := LocaleLowerCase('Hello World');  // X is 'hello world'
```

9.90 LocaleSameStr

Unit: Internal

Available In: Client and Server Applications

```
function LocaleSameStr(const A, B: String): Boolean
```

The **LocaleSameStr** function compares the A string input parameter with the B string input parameter with case-sensitivity. The comparison uses the browser's current locale setting to compare the two strings. The return value is a Boolean value of True if A is equal to B, and False if A is not equal to B.

Examples

```
X := LocaleSameStr('Absolute', 'Baseball');  // X is False
```

9.91 LocaleSameText

Unit: Internal

Available In: Client and Server Applications

```
function LocaleSameText(const A, B: String): Boolean
```

The **LocaleSameText** function compares the A string input parameter with the B string input parameter without case-sensitivity. The comparison uses the browser's current locale setting to compare the two strings. The return value is a Boolean value of True if A is equal to B, and False if A is not equal to B.

Examples

```
X := LocaleSameStr('Absolute', 'ABSOLUTE');  // X is True
```

9.92 LocaleUpperCase

Unit: Internal

Available In: Client and Server Applications

```
function LocaleUpperCase(const Value: String): String
```

The **LocaleUpperCase** function returns the Value input parameter with all characters converted to their upper-case representation. The browser's current locale setting is used to perform this conversion. The return value is a String value.

Examples

```
X := LocaleUpperCase('Hello World');  // X is 'HELLO WORLD'
```

9.93 LowerCase

Unit: Internal

Available In: Client and Server Applications

```
function LowerCase(const Value: String): String
```

The **LowerCase** function returns the Value input parameter with all characters converted to their lower-case representation. The browser's current locale setting is not used to perform this conversion. The return value is a String value.

Examples

```
X := LowerCase('Hello World');  // X is 'hello world'
```

9.94 Max

Unit: Internal

Available In: Client and Server Applications

```
function Max(A,B: Integer): Integer

function Max(A,B: Double): Double
```

The **Max** function returns the greater of the two input parameters. If A is greater than B, then A is returned. If B is greater than A, then B is returned. The return value is the same type as the input parameters.

Examples

```
X := Max(100,2);  // X is 100
```

9.95 MessageDlg

Unit: WebForms

Available In: Client Applications

```
procedure MessageDlg(const Msg: String;
                     const DlgCaption: String;
                     DlgType: TMsgDlgType;
                     const Buttons: TMsgDlgBtns;
                     MsgDlgResult: TMsgDlgResultEvent=nil;
                     CloseButton: Boolean=False;
                     OverlayColor: TColor=clBlack;
                     OverlayOpacity: Double=20;
                     AnimationStyle: TAnimationStyle=asNone;
                     AnimationDuration: Integer=0)

procedure MessageDlg(const Msg: String;
                     const DlgCaption: String;
                     DlgType: TMsgDlgType;
                     const Buttons: TMsgDlgBtns;
                     DefaultButton: TMsgDlgBtn;
                     MsgDlgResult: TMsgDlgResultEvent=nil;
                     CloseButton: Boolean=False;
                     OverlayColor: TColor=clBlack;
                     OverlayOpacity: Double=20;
                     AnimationStyle: TAnimationStyle=asNone;
                     AnimationDuration: Integer=0)
```

The **MessageDlg** procedure shows a modal message dialog.

The Msg parameter indicates the message to show.

The DlgCaption parameter indicates the caption of the dialog.

The DlgType parameter indicates the type of dialog to show.

The Buttons parameter indicates the array of button types to use on the dialog.

The DefaultButton parameter indicates which button should have focus when the dialog is first shown.

The MsgDlgResult parameter is a method reference that represents the event handler that will be called when the message dialog is closed.

The CloseButton parameter indicates whether the message dialog should contain a close button.

The OverlayColor parameter indicates the color to use for the shadow overlay effect that is used over the application desktop to indicate that a modal form is in effect.

The OverlayOpacity parameter indicates the percentage of opacity to use for the shadow overlay effect that is used over the application desktop to indicate that a modal form is in effect.

The AnimationStyle and AnimationDuration parameters indicate the type/duration of animation to use when showing the message dialog.

Examples

```
MessageDlg('Are you sure that you want to delete this record?',
           'Please Confirm',mtConfirmation,[mbYes,mbNo],mbNo,CheckDelete,
      True);
```

9.96 Min

Unit: Internal

Available In: Client and Server Applications

```
function Min(A,B: Integer): Integer

function Min(A,B: Double): Double
```

The **Min** function returns the lesser of the two input parameters. If A is less than B, then A is returned. If B is less than A, then B is returned. The return value is the same type as the input parameters.

Examples

```
X := Min(100,2);  // X is 2
```

9.97 MinuteOf

Unit: Internal

Available In: Client and Server Applications

```
function MinuteOf(Value: DateTime; UTC: Boolean=False): Integer
```

The **MinuteOf** function returns the minute number of the input parameter in local or UTC time. The return value is an Integer value between 0 and 59.

Examples

```
X := MinuteOf(Time);  // X is 10 (assuming a time of 12:10 PM)
```

9.98 MonthOf

Unit: Internal

Available In: Client and Server Applications

```
function MonthOf(Value: DateTime; UTC: Boolean=False): Integer
```

The **MonthOf** function returns the month number of the input parameter in local or UTC time. The return value is an Integer value.

Examples

```
X := MonthOf(Date);  // X is 2 (assuming a date of 02/13/2012)
```

9.99 MSecondOf

Unit: Internal

Available In: Client and Server Applications

```
function MSecondOf(Value: DateTime; UTC: Boolean=False): Integer
```

The **MSecondOf** function returns the millisecond number of the input parameter in local or UTC time. The return value is an Integer value between 0 and 59.

Examples

```
X := MSecondOf(Time);  // X is 247 (assuming a time of 12:10:20.247 PM)
```

9.100 Now

Unit: Internal

Available In: Client and Server Applications

```
function Now: DateTime
```

The **Now** function returns the current date and time. The return value is a DateTime value.

Examples

```
X := DateTimeToStr(Now);  // X is '2/13/2012 12:10 PM'
```

9.101 Ord

Unit: Internal

Available In: Client and Server Applications

```
function Ord(Value: Char): Integer

function Ord(Value: Boolean): Integer
```

The **Ord** function returns the Unicode code point for a Char input parameter, 0 for a False Boolean input parameter, and 1 for a True Boolean input parameter. The return value is an Integer.

Examples

```
X := Ord('Ü');  // X is 220

X := Ord(True);  // X is 1
```

9.102 Pad

Unit: WebCore

Available In: Client and Server Applications

```
function Pad(const Value: String; PadLen: Integer;
             PadChar: Char=' '): String
```

The **Pad** function returns the Value input parameter padded to the length specified by the PadLen input parameter. The optional PadChar input parameter specifies the character to be used for padding the Value input parameter, and defaults to a space (' ') character. The return value is a String value.

> **Note**
> The padding is inserted on the left side of the string.

Examples

```
X := Pad('100', 10);  // X is '       100'
```

9.103 Pi

Unit: Internal

Available In: Client and Server Applications

```
function Pi: Double
```

The **Pi** function returns the mathematical constant pi, or the ratio of the circumference of a circle to its diameter. The return value is a Double value that is approximately 3.141592653589793.

Examples

```
X := Pi;  // X is 3.141592653589793
```

9.104 ParseXML

Unit: WebCore

Available In: Client Applications

```
function ParseXML(const Value: String): TDocument
```

The **ParseXML** function parses the XML string input parameter and returns a TDocument class instance.

> **Note**
> Please refer to the WebDOM unit source code for the declaration of the TNode, TDocument, and TNodeList classes and their various properties.

Examples

```
var
   TempXML: String;
   TempDocument: TDocument;
   TempNodes: TNodeList;
begin
   TempXML := '<a><b><c><username>testuser</username></c></b></a>';
   TempDocument := ParseXML(TempXML);
   TempNodes := TempDocument.getElementsByTagName('username');
   ShowMessage(IntToStr(TempNodes.length)); // Number of nodes
   ShowMessage(TempNodes[0].firstChild.nodeValue); // Get text node
   TempXML := SerializeXML(TempDocument);
   ShowMessage(TempXML);
end
```

9.105 Pos

Unit: Internal

Available In: Client and Server Applications

```
function Pos(const SearchValue: String; const Value: String;
             Index: Integer=1): Integer
```

The **Pos** function returns the position of the SearchValue input parameter in the Value input parameter. The optional Index parameter specifies the starting index of the search and, if not specified, the search will start at the first character of the Value input parameter. The return value is an Integer value of 0 if the SearchValue input parameter was not found, or the index of the SearchValue if it was found.

Examples

```
X := Pos(' ', 'Whereisthe spaceinthisstring');  // X is 11
```

9.106 Power

Unit: Internal

Available In: Client and Server Applications

```
function Power(X, Y: Double): Double

function Power(X, Y: Integer): Double
```

The **Power** function returns the X input parameter raised to the power specified by the Y input parameter. The return value is a Double value.

Examples

```
X := Power(0.523, 4);  // X is 0.07481811384100001
```

9.107 Radians

Unit: Internal

Available In: Client and Server Applications

```
function Radians(Value: Double): Double

function Radians(Value: Integer): Double
```

The **Radians** function converts the input parameter, which is an angle specified in degrees, to radians. The return value is a Double value.

Examples

```
X := Radians(5294.01543544978);  // X is 92.398
```

9.108 Random

Unit: Internal

Available In: Client and Server Applications

```
function Random(AFrom: Integer=0; ATo: Integer=<MaxInt>): Integer
```

The **Random** function returns a pseudorandom number greater than or equal to the AFrom parameter, if provided, and less than or equal to the ATo parameter, if provided. The default AFrom parameter value is 0, and the default ATo parameter value is the maximum integer value. At runtime, the maximum integer value is 9007199254740991, and at design-time the maximum integer value is 9223372036854775807. The return value is an Integer value.

Examples

```
X := Random;  // X is 7534176611 (pseudorandom value)
X := Random(0,1000);  // X is 269 (pseudorandom value)
```

9.109 RemoveDirectory

Unit: Internal

Available In: Server Applications

```
procedure RemoveDirectory(const Directory: String)
```

The **RemoveDirectory** procedure removes the directory specified in the input parameter. The specified directory path can contain intermediate directories, but those intermediate directories must exist and the procedure will only remove the final directory in the path. If the directory cannot be removed for any reason, an exception will be raised.

> **Note**
> The specified directory path should be an absolute path. Relative paths are permitted, but they rely on the current working directory. The current working directory is a per-process setting and is not reliable in a multi-threaded setting like the web server.

Examples

```
if DirectoryExists('C:\temp\backup') then
   RemoveDirectory('C:\temp\backup');
```

9.110 RemoveTrailingPathDelim

Unit: WebCore

Available In: Client and Server Applications

```
function RemoveTrailingPathDelim(const Value: String;
                                 Delimiter: String=SLASH): String
```

The **RemoveTrailingPathDelim** function removes the specified path delimiter from the end of the string input parameter, if the path delimiter exists at the end of the string. The return value is a string.

Examples

```
X := RemoveTrailingPathDelim('https://www.elevatesoft.com/');  // X is
      'https://www.elevatesoft.com'
```

9.111 Round

Unit: Internal

Available In: Client and Server Applications

```
function Round(Value: Double): Integer

function Round(Value: Integer): Integer
```

The **Round** function returns the closest integer to the value of the input parameter using the "round half up" method. The return value is an Integer.

Examples

```
X := Round(-10.4);  // X is -10
X := Round(15.5);  // X is 16
```

9.112 QuotedStr

Unit: WebCore

Available In: Client and Server Applications

```
function QuotedStr(const Value: String;
                   QuoteChar: Char=SINGLE_QUOTE): String
```

The **QuotedStr** function adds the specified quote character to the start and end of the input parameter. In addition, any characters in the input parameter that match the specified quote character are escaped so that they are properly interpreted as embedded quote characters. The return value is the transformed input parameter.

Examples

```
Y := 'Absolute';
X := QuotedStr(Y);  // X is 'Absolute'

Y := 'It''s';
X := QuotedStr(Y);  // X is 'Its''s'
```

9.113 SameStr

Unit: Internal

Available In: Client and Server Applications

```
function SameStr(const A, B: String): Boolean
```

The **SameStr** function compares the A string input parameter with the B string input parameter with case-sensitivity. The comparison is locale-insensitive. The return value is a Boolean value of True if A is equal to B, and False if A is not equal to B.

Examples

```
X := SameStr('Absolute', 'Baseball');  // X is False
```

9.114 SameText

Unit: Internal

Available In: Client and Server Applications

```
function SameText(const A, B: String): Boolean
```

The **SameText** function compares the A string input parameter with the B string input parameter without case-sensitivity. The comparison is locale-insensitive. The return value is a Boolean value of True if A is equal to B, and False if A is not equal to B.

Examples

```
X := SameStr('Absolute', 'ABSOLUTE');  // X is True
```

9.115 SecondOf

Unit: Internal

Available In: Client and Server Applications

```
function SecondOf(Value: DateTime; UTC: Boolean=False): Integer
```

The **SecondOf** function returns the second number of the input parameter in local or UTC time. The return value is an Integer value between 0 and 59.

Examples

```
X := SecondOf(Time);  // X is 20 (assuming a time of 12:10:20 PM)
```

9.116 SerializeXML

Unit: WebCore

Available In: Client Applications

```
function SerializeXML(Document: TDocument): String
```

The **SerializeXML** function converts the XML nodes present in the TDocument instance parameter into a string. The return value is a String value.

> **Note**
> Please refer to the WebDOM unit source code for the declaration of the TNode, TDocument, and TNodeList classes and their various properties.

Examples

```
var
   TempXML: String;
   TempDocument: TDocument;
   TempNodes: TNodeList;
begin
   TempXML := '<a><b><c><username>testuser</username></c></b></a>';
   TempDocument := ParseXML(TempXML);
   TempNodes := TempDocument.getElementsByTagName('username');
   ShowMessage(IntToStr(TempNodes.length)); // Number of nodes
   ShowMessage(TempNodes[0].firstChild.nodeValue); // Get text node
   TempXML := SerializeXML(TempDocument);
   ShowMessage(TempXML);
end
```

9.117 SetLength

Unit: Internal

Available In: Client and Server Applications

```
procedure SetLength(var Value: array of <Type>; Length: Integer)
```

The **SetLength** procedure sets the length of the Value array input parameter. If extending the length of an array, all new elements are automatically set to nil for string, object, or method arrays, NaN (not a number) for numeric (Integer or Double) arrays, and False for boolean arrays.

> **Warning**
> Arrays that are declared but not assigned a value are nil (Assigned function returns False) and not initialized. The SetLength procedure is one way of initializing them, even if they are initialized to a length of 0.

Examples

```
var
   X: array of String;
   I: Integer;
begin
   SetLength(X, 10);  // X now has a length of 10, but each element is still
      nil
   for I := 0 to Length(X)-1 do
      X[I] := '';  //  Initialize each array element with an empty string
end
```

9.118 ShowMessage

Unit: WebForms

Available In: Client Applications

```
procedure ShowMessage(const Msg: String;
                      const DlgCaption: String='';
                      AnimationStyle: TAnimationStyle=asNone;
                      AnimationDuration: Integer=0)
```

The **ShowMessage** procedure shows a simple modal message dialog. The Msg parameter indicates the message to show. The DlgCaption parameter is optional and indicates the caption of the dialog. The AnimationStyle and AnimationDuration parameters are optional, and indicate the type/duration of animation to use when showing the message dialog.

Examples

```
ShowMessage('An error has occurred !','Error');

ShowMessage('Hello world !','Hi !',asQuadEaseOut,500);
```

9.119 ShowProgress

Unit: WebForms

Available In: Client Applications

```
procedure ShowProgress(const Msg: String;
                       AnimationStyle: TAnimationStyle=asNone;
                       AnimationDuration: Integer=0)
```

The **ShowProgress** procedure shows a modal progress dialog and increments the global progress reference count. The HideProgress procedure decrements the reference count and hides any active progress dialog. The AnimationStyle and AnimationDuration parameters are optional, and indicate the type/duration of animation to use when showing the progress dialog.

Examples

```
ShowProgress('Loading customers...');
```

9.120 Sin

Unit: Internal

Available In: Client and Server Applications

```
function Sin(Value: Double): Double

function Sin(Value: Integer): Double
```

The **Sin** function returns the sine of the input parameter, which is an angle specified in radians. To convert an angle from degrees to radians, use the Radians function. The return value is a Double value between -1 and 1.

Examples

```
X := Sin(0.23290);  // X is 0.2308001934780994
```

9.121 Split

Unit: Internal

Available In: Client and Server Applications

```
function Split(const Value: String; const Separator: String): array of String

function Split(const Value: String; const Separator: String;
               MaxLength: Integer): array of String
```

The **Split** function builds a new string array by splitting the Value input parameter using the Separator input parameter. The optional MaxLength input parameter specifies the maximum length of the string array. The return value is an array of String values that **does not** include the specified Separator string.

Examples

```
X := Split('Hello, my name is Jim', ' ');  // X is ['Hello,','my','name','is',
      'Jim']
```

9.122 Sqrt

Unit: Internal

Available In: Client and Server Applications

```
function Sqrt(Value: Double): Double

function Sqrt(Value: Integer): Double
```

The **Sqrt** function returns the square root of the input parameter. The return value is a Double value.

Examples

```
X := Sqrt(154);  // X is 12.409673645990857
```

9.123 StrReplace

Unit: WebCore

Available In: Client and Server Applications

```
function StrReplace(const Value: String; const SearchValue: String;
                    const ReplaceValue: String;
                    ReplaceAll: Boolean=False;
                    CaseInsensitive: Boolean=False): String
```

The **StrReplace** function searches for the SearchValue input parameter in the Value input parameter and replaces it with the ReplaceValue input parameter. If the optional ReplaceAll input parameter is True, then all occurrences of the SearchValue input parameter are replaced with the ReplaceValue input parameter. If the optional CaseInsensitive input parameter is True, then the search for the SearchValue input parameter will be case-insensitive. The return value is the modified String value.

Examples

```
X := StrReplace('abcdefghijk', 'd', ' ', True, True);  // X is 'abc efghijk'
```

9.124 StrToBool

Unit: WebCore

Available In: Client and Server Applications

```
function StrToBool(const Value: String): Boolean
```

The **StrToBool** function returns True if the input parameter is 'True' (case-insensitive), and False if the input parameter is 'False' (case-insensitive also). The return value is a Boolean value.

Examples

```
X := StrToBool('True');  // X is True
```

9.125 StrToBoolDef

Unit: WebCore

Available In: Client and Server Applications

```
function StrToBoolDef(const Value: String;
                      DefaultValue: Boolean=False): Boolean
```

The **StrToBoolDef** function returns True if the input parameter is 'True' (case-insensitive), and False if the input parameter is 'False' (case-insensitive also). If the input parameter is not a valid boolean string, the return value is the specified default value.

Examples

```
X := StrToBoolDef('Hello',False);  // X is False
```

9.126 StrToDate

Unit: WebCore

Available In: Client and Server Applications

```
function StrToDate(const Value: String; UTC: Boolean=False): DateTime
```

The **StrToDate** function converts the formatted local or UTC date string input parameter into its native value. The required format of the string is determined by the TFormatSettings ShortDateFormat property. The return value is a DateTime value.

Examples

```
A := StrToDate('2/13/2012');
X := DateToStr(A);  // X is '2/13/2012'
```

9.127 StrToDateDef

Unit: WebCore

Available In: Client and Server Applications

```
function StrToDateDef(const Value: String; UTC: Boolean=False;
                      DefaultValue: DateTime=DateTime(0)): DateTime
```

The **StrToDateDef** function converts the formatted local or UTC date string input parameter into its native value. The required format of the string is determined by the TFormatSettings ShortDateFormat property. If the input parameter is not a valid date string, the return value is the specified default value.

Examples

```
A := StrToDateDef('2/33/2012', StrToDate('1/1/2012'));
X := DateToStr(A);  // X is '1/1/2012'
```

9.128 StrToDateTime

Unit: WebCore

Available In: Client and Server Applications

```
function StrToDateTime(const Value: String; UTC: Boolean=False): DateTime
```

The **StrToDateTime** function converts the formatted local or UTC date and time string input parameter into its native value. The required format of the string is determined by the TFormatSettings ShortDateFormat and ShortTimeFormat properties. The return value is a DateTime value.

Examples

```
A := StrToDateTime('2/13/2012 12:10 PM');
X := DateTimeToStr(A);  // X is '2/13/2012 12:10 PM'
```

9.129 StrToDateTimeDef

Unit: WebCore

Available In: Client and Server Applications

```
function StrToDateTimeDef(const Value: String; UTC: Boolean=False;
                          DefaultValue: DateTime=DateTime(0)): DateTime
```

The **StrToDateTimeDef** function converts the formatted local or UTC date and time string input parameter into its native value. The required format of the string is determined by the TFormatSettings ShortDateFormat and ShortTimeFormat properties. If the input parameter is not a valid date and time string, the return value is the specified default value.

Examples

```
A := StrToDateTimeDef('2/33/2012 12:10 PM', StrToDateTime('1/1/2012 12:00
     AM'));
X := DateTimeToStr(A);  // X is '1/1/2012 12:00 AM'
```

9.130 StrToDouble

Unit: Internal

Available In: Client and Server Applications

```
function StrToDouble(const Value: String): Double
```

The **StrToDouble** function converts the formatted string input parameter into its native value. The decimal separator used in the formatted string is **always** required to be a period (.). The return value is a Double value.

Examples

```
A := StrToDouble('1200.548');  // X is 1200.548
```

9.131 StrToFloat

Unit: Internal

Available In: Client and Server Applications

```
function StrToFloat(const Value: String): Double
```

The **StrToFloat** function converts the formatted string input parameter into its native value. The decimal separator used in the formatted string is determined by the TFormatSettings DecimalSeparator property. The return value is a Double value.

Examples

```
A := StrToFloat('1200.548');
X := FloatToStr(A);  // X is '1200.548'
```

9.132 StrToFloatDef

Unit: Internal

Available In: Client and Server Applications

```
function StrToFloatDef(const Value: String;
                       DefaultValue: Double=0): Double
```

The **StrToFloatDef** function converts the formatted string input parameter into its native value. The decimal separator used in the formatted string is determined by the TFormatSettings DecimalSeparator property. If the input parameter is not a valid floating-point string, the return value is the specified default value.

Examples

```
A := StrToFloatDef('1200.548', 0);
X := FloatToStr(A);  // X is '1200.548'
```

9.133 StrToInt

Unit: Internal

Available In: Client and Server Applications

```
function StrToInt(const Value: String): Integer
```

The **StrToInt** function converts the formatted string input parameter into its native value. The return value is an Integer value.

Examples

```
X := StrToInt('-102');  // X is -102
```

9.134 StrToIntDef

Unit: Internal

Available In: Client and Server Applications

```
function StrToIntDef(const Value: String;
                     DefaultValue: Integer=0): Integer
```

The **StrToIntDef** function converts the formatted string input parameter into its native value. If the input parameter is not a valid integer string, the return value is the specified default value.

Examples

```
X := StrToIntDef('-102', 0);  // X is -102
```

9.135 StrToTime

Unit: WebCore

Available In: Client and Server Applications

```
function StrToTime(const Value: String; UTC: Boolean=False): DateTime
```

The **StrToTime** function converts the formatted local or UTC time string input parameter into its native value. The required format of the string is determined by the TFormatSettings ShortTimeFormat properties. The return value is a DateTime value.

Examples

```
A := StrToTime('12:10 PM');
X := TimeToStr(A);  // X is '12:10 PM'
```

9.136 StrToTimeDef

Unit: WebCore

Available In: Client and Server Applications

```
function StrToTimeDef(const Value: String; UTC: Boolean=False;
                      DefaultValue: DateTime=DateTime(0)): DateTime
```

The **StrToTimeDef** function converts the formatted local or UTC time string input parameter into its native value. The required format of the string is determined by the TFormatSettings ShortTimeFormat properties. If the input parameter is not a valid time string, the return value is the specified default value.

Examples

```
A := StrToTimeDef('12:10 PM', TimeToStr('12:00 AM'));
X := TimeToStr(A);  // X is '12:10 PM'
```

9.137 Tan

Unit: Internal

Available In: Client and Server Applications

```
function Tan(Value: Double): Double

function Tan(Value: Integer): Double
```

The **Tan** function returns the tangent of the input parameter, which is an angle specified in radians. To convert an angle from degrees to radians, use the Radians function. The return value is a Double value.

Examples

```
X := Tan(0.23290);  // X is 0.23720443648121617
```

9.138 TempDirectory

Unit: Internal

Available In: Server Applications

```
function TempDirectory: String
```

The **TempDirectory** function returns the temporary files directory in the operating system for the current user account.

Examples

```
X := TempDirectory;  // X is 'C:\Users\User\AppData\Local\Temp\'
```

9.139 TempFileName

Unit: Internal

Available In: Server Applications

```
function TempFileName(const Directory: String=''): String
```

The **TempFileName** function returns a unique temporary file name for the directory input parameter. If the directory input parameter is blank (''), then the operating system temporary files directory will be used for generating the temporary file name. The return value is a string with the following format:

```
ewb<Unique 2-digit Hex Number>.tmp
```

Note
The returned file name is guaranteed to be unique within the applicable directory among all server applications running in the web server, but not guaranteed to be unique outside of that context.

Examples

```
X := TempFileName;  // X is 'C:\Users\User\AppData\Local\Temp\ewbCACF.tmp'
```

9.140 Time

Unit: Internal

Available In: Client and Server Applications

```
function Time: DateTime
```

The **Time** function returns the current time. The return value is a DateTime value.

> **Note**
> DateTime values encompass both the date and time portion for a given date/time. The Time function will set the date portion of the date/time to the current date.

Examples

```
X := TimeToStr(Time);  // X is '12:10 PM'
```

9.141 TimeToStr

Unit: WebCore

Available In: Client and Server Applications

```
function TimeToStr(Value: DateTime; UTC: Boolean=False): String
```

The **TimeToStr** function returns a formatted local or UTC time string for the DateTime input parameter. The format of the string is determined by the TFormatSettings ShortTimeFormat properties. The return value is a String value.

Examples

```
A := StrToTime('12:10 PM');
X := TimeToStr(A);  // X is '12:10 PM'
```

9.142 Trim

Unit: WebCore

Available In: Client and Server Applications

```
function Trim(const Value: String): String

function Trim(const Value: String; TrimChar: Char): String
```

The **Trim** function returns the Value input parameter with both leading and trailing "space" characters removed. The first version of this function trims all leading and trailing characters that are less than or equal to the space (#32) character from the string. The second version of this function allows the developer to specify the character that should be trimmed from the Value parameter. The return value is a String value.

Examples

```
X := Trim('  Hello World  ');  // X is 'Hello World'
```

9.143 TimeZoneOffset

Unit: Internal

Available In: Client and Server Applications

```
function TimeZoneOffset(Value: DateTime): Integer
```

The **TimeZoneOffset** function returns the time zone offset for the input parameter. The return value is an Integer value that represents the time zone offset expressed in minutes.

Examples

```
X := TimeZoneOffset(Now);  // X is 240 (4 hours) for US EST during
                                  // daylight savings time (summer)
```

9.144 Trunc

Unit: WebCore

Available In: Client and Server Applications

```
function Trunc(Value: Double): Integer

function Trunc(Value: Integer): Integer
```

The **Trunc** function returns the closest (towards 0) integer from the value of the input parameter. The return value is an Integer.

Examples

```
X := Trunc(-10.4);  // X is -10
X := Trunc(15.98);  // X is 15
```

9.145 UnEscapeJSON

Unit: WebCore

Available In: Client and Server Applications

```
function UnEscapeJSON(const Value: String): String
```

The **UnEscapeJSON** function returns the un-escaped version of the escaped JSON input parameter. The return value is a String value.

This function is the analog of the EscapeJSON function.

Examples

```
X := UnEscapeJSON('\"Hello World\"');  // X is '"Hello World"'
```

9.146 UpperCase

Unit: Internal

Available In: Client and Server Applications

```
function UpperCase(const Value: String): String
```

The **UpperCase** function returns the Value input parameter with all characters converted to their upper-case representation. The browser's current locale setting is not used to perform this conversion. The return value is a String value.

Examples

```
X := UpperCase('Hello World');  // X is 'HELLO WORLD'
```

9.147 VarClear

Unit: Internal

Available In: Client and Server Applications

```
procedure VarClear(var Value: Variant)
```

The **VarClear** procedure sets the variant input parameter to Null.

> **Note**
> The Null value is a special variant value that indicates that a given variant does not have a value (or type). It is different from a nil value that is used with object instances, class types, method and routine references, and arrays.

Examples

```
var
   MyValue: Variant;
begin
   MyValue := 'Hello World';
   X := VarNull(MyValue);  // X is False
   VarClear(MyValue);
   X := VarNull(MyValue);  // X is True
end
```

9.148 VarNull

Unit: Internal

Available In: Client and Server Applications

```
function VarNull(const Value: Variant)
```

The **VarNull** function returns True if the variant input parameter is Null and False if the variant input parameter has been assigned a value.

> **Note**
> The Null value is a special variant value that indicates that a given variant does not have a value (or type). It is different from a nil value that is used with object instances, class types, method and routine references, and arrays.

Examples

```
var
   MyValue: Variant;
begin
   MyValue := 'Hello World';
   X := VarNull(MyValue);  // X is False
   VarClear(MyValue);
   X := VarNull(MyValue);  // X is True
end
```

9.149 VarType

Unit: Internal

Available In: Client and Server Applications

```
function VarType(const Value: Variant)
```

The **VarType** function returns an integer value representing the type of the value, if one is present, in the variant input parameter. The following variant types are defined as constants in the **WebCore** unit:

Type	Description
VARTYPE_NULL (0)	Null (no value)
VARTYPE_ARRAY (1)	Array
VARTYPE_BOOLEAN (2)	Boolean
VARTYPE_CHAR (3)	Char
VARTYPE_CLASS (4)	Class
VARTYPE_DOUBLE (5)	Double
VARTYPE_EXTCLASS (6)	External Class
VARTYPE_EXTINSTANCE (7)	External Instance
VARTYPE_EXTMETHOD (8)	External Method
VARTYPE_INSTANCE (9)	Instance
VARTYPE_INTEGER (10)	Integer
VARTYPE_METHOD (11)	Method
VARTYPE_ROUTINE (12)	Routine
VARTYPE_STRING (13)	String

Examples

```
var
   MyValue: Variant;
begin
   MyValue := 'Hello World';
   X := VarType(MyValue);  // X is 13
   MyValue := 198;
   X := VarType(MyValue);  // X is 10
end
```

9.150 WeekDayOf

Unit: Internal

Available In: Client and Server Applications

```
function WeekDayOf(Value: DateTime; UTC: Boolean=False): Integer
```

The **WeekDayOf** function returns the week day number of the input parameter in local or UTC time. This function is ISO 8601-compliant, meaning that the week days of Monday through Sunday are represented by the values 1 through 7, respectively. The return value is an Integer value.

Examples

```
X := WeekDayOf(Date);  // X is 1 (Monday, assuming a date of 02/13/2012)
```

9.151 YearOf

Unit: Internal

Available In: Client and Server Applications

```
function YearOf(Value: DateTime; UTC: Boolean=False): Integer
```

The **YearOf** function returns the year number of the input parameter in local or UTC time. The return value is an Integer value.

Examples

```
X := YearOf(Date);  // X is 2012 (assuming a date of 02/13/2012)
```

Chapter 10
Component Reference

10.1 TAbstractList Component

Unit: WebCore

Inherits From TPersistent

Available In: Client and Server Applications

This class represents an abstract list and is used as the ancestor class for the TObjectList and TStrings classes. It provides the functionality for tracking changes to the list as well as dealing with batch updates to the list.

Properties	Methods	Events
	BeginUpdate	OnChanged
	EndUpdate	

TAbstractList.BeginUpdate Method

```
procedure BeginUpdate
```

Available In: Client and Server Applications

Use this method to begin a batch update to the list. Batch updates are useful in situations where many changes need to be made to the list, and triggering the OnChanged event on every change would result in performance issues. This method is reference-counted and every time it is called, an internal counter is incremented. Every time the EndUpdate method is called, the counter is decremented. Once the counter reaches zero, the OnChanged event will be triggered.

TAbstractList.EndUpdate Method

```
procedure EndUpdate
```

Available In: Visual Client Applications

Use this method to end a batch update to the list. Batch updates are useful in situations where many changes need to be made to the list, and triggering the OnChanged event on every change would result in performance issues. This method is reference-counted and every time it is called, an internal counter is decremented. Every time the BeginUpdate method is called, the counter is incremented. Once the counter reaches zero, the OnChanged event will be triggered.

TAbstractList.OnChanged Event

```
property OnChanged: TNotifyEvent
```

Available In: Client and Server Applications

This event is triggered whenever the list is modified in any way. If a batch update is in effect via the BeginUpdate and EndUpdate methods, then this event is only triggered once the EndUpdate call is made. If multiple calls to BeginUpdate and EndUpdate are nested, then this event is only triggered once the last matching EndUpdate call is made.

10.2 TAddress Component

Unit: WebComps

Inherits From TObject

Available In: Client Applications

The TAddress class encapsulates the address bar (location) functionality in the web browser.

> **Note**
> The component library includes a global instance variable of this class called Address in the WebComps unit that should be used instead of creating new instances of the class.

> **Warning**
> The methods of this class, as well as assignments to the various properties, can cause the browser to navigate to a new resource location, and this will cause the current application to be unloaded. The sole exception are any assignments to the Anchor property. Changes to the Anchor property will not cause the current application to be unloaded.

Properties	Methods	Events
Anchor	Assign	OnAnchorChange
Host	Create	
HostName	Reload	
Params	Replace	
Path		
Port		
Protocol		
URL		

TAddress.Anchor Property

```
property Anchor: String
```

Available In: Client Applications

Specifies the anchor (#) portion for the address. If the specified anchor is different than the anchor in the current address, then the browser will trigger the event handler assigned to the OnAnchorChange event property of the global Address instance. The assigned event handler can then take specific action based upon the change.

TAddress.Host Property

```
property Host: String
```

Available In: Client Applications

Specifies the host (host name and port) portion for the address. If the specified host is different than the host in the current address, then the browser will unload the current application and load the resource specified by the new composed address.

TAddress.HostName Property

```
property HostName: String
```

Available In: Client Applications

Specifies the host name (www.mysite.com) portion for the address. If the specified host name is different than the host name in the current address, then the browser will unload the current application and load the resource specified by the new composed address.

TAddress.Params Property

```
property Params: String
```

Available In: Client Applications

Specifies the query parameters (?param1=100&param2=200) portion for the address. If the specified query parameters are different than the query parameters in the current address, then the browser will unload the current application and load the resource specified by the new composed address.

TAddress.Path Property

```
property Path: String
```

Available In: Client Applications

Specifies the path portion for the address. If the specified path is different than the path in the current address, then the browser will unload the current application and load the resource specified by the new composed address.

TAddress.Port Property

```
property Port: String
```

Available In: Client Applications

Specifies the port (:80, :8080, etc.) portion for the address. If the specified port is different than the port in the current address, then the browser will unload the current application and load the resource specified by the new composed address.

TAddress.Protocol Property

```
property Protocol: String
```

Available In: Client Applications

Specifies the protocol (normally http: or https:) portion for the address. If the specified protocol is different than the protocol in the current address, then the browser will unload the current application and load the resource specified by the new composed address.

TAddress.URL Property

```
property URL: String
```

Available In: Client Applications

Specifies the complete address. If the specified address is different than the current address, then the browser will unload the current application and load the resource specified by the new address.

TAddress.Assign Method

```
procedure Assign(const AURL: String)
```

Available In: Client Applications

Use this method to load the resource specified by the AURL parameter. Using this method will result in a new history entry for the resource in the browser's navigation history.

TAddress.Create Method

```
constructor Create
```

Available In: Client Applications

Use this method to create a new instance of the TAddress class.

TAddress.Reload Method

```
procedure Reload(Force: Boolean=False)
```

Available In: Client Applications

Use this method to reload the resource for the current address. By default, the browser will attempt to reload the resource from its cache. Setting the Force parameter to True will cause the browser to ignore its cached and reload the resource from its source (web server, file system, etc.).

TAddress.Replace Method

```
procedure Replace(const AURL: String)
```

Available In: Client Applications

Use this method to load the resource specified by the AURL parameter. Using this method will result in the history entry for the current address being replaced in the browser's navigation history with the address of the resource.

TAddress.OnAnchorChange Event

```
property OnAnchorChange: TNotifyEvent
```

Available In: Client Applications

This event is triggered whenever the anchor (#) portion of the current address changes. This is useful for navigating in your application using the browser's address bar and forward/back buttons.

10.3 TAlertLabel Component

Unit: WebLabels

Inherits From TAlertLabelControl

Available In: Visual Client Applications

The TAlertLabel component represents a label control that can be used to display alerts that, by default, appear at the top of the client area of their container, with customizations such as the orientation of the caption and whether or not to show a close button.

Properties	Methods	Events
AllowClose		OnAnimationComplete
AutoHeight		OnAnimationsComplete
Background		OnCaptureEnd
Border		OnCaptureStart
Caption		OnCapturing
Corners		OnClick
Cursor		OnClose
DataColumn		OnCloseQuery
DataSet		OnContextMenu
Font		OnDblClick
Format		OnHide
Hint		OnMouseDown
Opacity		OnMouseEnter
Orientation		OnMouseLeave
Padding		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TAlertLabel.AllowClose Property

```
property AllowClose: Boolean
```

Available In: Visual Client Applications

Specifies whether the close button should be shown.

TAlertLabel.AutoHeight Property

```
property AutoHeight: Boolean
```

Available In: Visual Client Applications

Specifies whether the height of the alert label should be automatically set based upon the Caption, Font, and Format properties.

TAlertLabel.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

TAlertLabel.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TAlertLabel.Caption Property

```
property Caption: TCaption
```

Available In: Visual Client Applications

Specifies the text to be shown in the alert label control. The text can contain line feeds. The default value is ''.

TAlertLabel.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TAlertLabel.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TAlertLabel.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TAlertLabel.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

TAlertLabel.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TAlertLabel.Format Property

```
property Format: TFormat
```

Available In: Visual Client Applications

Specifies the content formatting to use for the control's Caption.

TAlertLabel.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TAlertLabel.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TAlertLabel.Orientation Property

```
property Orientation: TAlertOrientation
```

Available In: Visual Client Applications

Specifies the orientation of the alert label caption.

TAlertLabel.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

TAlertLabel.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TAlertLabel.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TAlertLabel.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TAlertLabel.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TAlertLabel.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TAlertLabel.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TAlertLabel.OnClose Event

```
property OnClose: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the label is closed by the user via the close button, or when the Close method is called.

TAlertLabel.OnCloseQuery Event

```
property OnCloseQuery: TCloseQueryEvent
```

Available In: Visual Client Applications

This event is triggered when the label is closed by the user via the close button, or when the Close method is called. Return True to allow the close to continue, or False to prevent the label from closing.

TAlertLabel.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TAlertLabel.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TAlertLabel.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TAlertLabel.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TAlertLabel.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TAlertLabel.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TAlertLabel.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TAlertLabel.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TAlertLabel.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TAlertLabel.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TAlertLabel.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TAlertLabel.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TAlertLabel.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TAlertLabel.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TAlertLabel.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.4 TAlertLabelControl Component

Unit: WebLabels

Inherits From TBindableColumnControl

Available In: Visual Client Applications

The TAlertLabelControl control is the base class for alert label controls, and contains all of the label functionality in the form of public methods and protected properties/events that descendant classes can use to create customized alert label controls.

Properties	Methods	Events
	Close	

TAlertLabelControl.Close Method

```
procedure Close
```

Available In: Visual Client Applications

Use this method to close the label. When this method is called, the OnCloseQuery event is triggered, followed by the OnClose event. If the OnCloseQuery event handler returns True, then the label will be hidden before the OnClose event is triggered.

10.5 TAnimatedIcon Component

Unit: WebIcons

Inherits From TIconControl

Available In: Visual Client Applications

The TAnimatedIcon component represents an animated icon control. An animated icon control displays a special type of icon, referenced by the Icon property, that contains a series of animation frames as a single background image. These animation frames can be oriented horizontally or vertically, and the Direction property allows you to specify the direction.

Properties	Methods	Events
Cursor	StartAnimating	OnAnimationComplete
Direction	StopAnimating	OnAnimationsComplete
Hint		OnCaptureEnd
Icon		OnCaptureStart
Opacity		OnCapturing
		OnClick
		OnContextMenu
		OnDblClick
		OnHide
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TAnimatedIcon.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TAnimatedIcon.Direction Property

```
property Direction: TAnimatedIconDirection
```

Available In: Visual Client Applications

Specifies the direction in which the animation frames in the icon are oriented.

TAnimatedIcon.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TAnimatedIcon.Icon Property

```
property Icon: TIconProperties
```

Available In: Visual Client Applications

Specifies the properties of the icon used with the control.

TAnimatedIcon.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TAnimatedIcon.StartAnimating Method

```
procedure StartAnimating
```

Available In: Visual Client Applications

Use this method to begin animating the icon specified in the Icon property.

TAnimatedIcon.StopAnimating Method

```
procedure StopAnimating
```

Available In: Visual Client Applications

Use this method to stop animating the icon specified in the Icon property.

TAnimatedIcon.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TAnimatedIcon.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TAnimatedIcon.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TAnimatedIcon.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TAnimatedIcon.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TAnimatedIcon.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TAnimatedIcon.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TAnimatedIcon.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TAnimatedIcon.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TAnimatedIcon.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TAnimatedIcon.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TAnimatedIcon.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TAnimatedIcon.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TAnimatedIcon.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TAnimatedIcon.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TAnimatedIcon.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TAnimatedIcon.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TAnimatedIcon.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TAnimatedIcon.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TAnimatedIcon.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TAnimatedIcon.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.6 TAnimation Component

Unit: WebUI

Inherits From TElementAttribute

Available In: Visual Client Applications

The TAnimation class represents the properties of an animation. The animation properties include the style of the animation and the duration, in milliseconds, of the animation.

Properties	Methods	Events
Duration	Cancel	
Running	SetToDefault	
Style	Start	

TAnimation.Duration Property

```
property Duration: Integer
```

Available In: Visual Client Applications

Specifies how long, in milliseconds, the animation should take to execute.

TAnimation.Running Property

```
property Running: Boolean
```

Available In: Visual Client Applications

Specifies if the animation is currently running.

TAnimation.Style Property

```
property Style: TAnimationStyle
```

Available In: Visual Client Applications

Specifies the style of the animation, which controls how the animation transforms a given UI element/control property. Currently, the supported styles include all of the standard easing transformations (including linear).

TAnimation.Cancel Method

```
procedure Cancel
```

Available In: Visual Client Applications

Use this method to cancel an animation.

Warning
Do not directly call this method. It is used internally by the interface manager.

TAnimation.SetToDefault Method

```
procedure SetToDefault
```

Available In: Visual Client Applications

Use this method to reset the animation's properties to their default values.

TAnimation.Start Method

```
procedure Start(EndValue: Integer)
```

Available In: Visual Client Applications

Use this method to start an animation.

Warning
Do not directly call this method. It is used internally by the interface manager.

10.7 TAnimations Component

Unit: WebUI

Inherits From TElementAttribute

Available In: Visual Client Applications

The TAnimations class represents the properties that can be animationed for a UI element or control. These properties currently include the Left, Top, Width, Height, Opacity, and Visible properties.

> **Note**
> When an animation is specified for a property, then that animation is applied **whenever** the property changes.

Properties	Methods	Events
Height	SetToDefault	
Left		
Opacity		
Top		
Visible		
Width		

TAnimations.Height Property

```
property Height: TAnimation
```

Available In: Visual Client Applications

Specifies the animation properties for the Height property.

TAnimations.Left Property

```
property Left: TAnimation
```

Available In: Visual Client Applications

Specifies the animation properties for the Left property.

TAnimations.Opacity Property

```
property Opacity: TAnimation
```

Available In: Visual Client Applications

Specifies the animation properties for the Opacity property.

TAnimations.Top Property

```
property Top: TAnimation
```

Available In: Visual Client Applications

Specifies the animation properties for the Top property.

TAnimations.Visible Property

```
property Visible: TAnimation
```

Available In: Visual Client Applications

Specifies the animation properties for the Visible property.

TAnimations.Width Property

```
property Width: TAnimation
```

Available In: Visual Client Applications

Specifies the animation properties for the Width property.

TAnimations.SetToDefault Method

```
procedure SetToDefault
```

Available In: Visual Client Applications

Use this method to reset all animation properties to their default values.

10.8 TApplication Component

Unit: WebRequest

Inherits From TComponent

Available In: Visual Client and Server Applications

The TApplication component represents the current instance of a visual client or server application.

For visual client applications, the TApplication component provides properties and methods for dealing with the browser viewport, application surface, forms, and global error handling.

For server applications, the TApplication component provides properties and methods for dealing with web server requests and sessions, web server request handlers, and global error handling.

An instance of the TApplication component called **Application** is automatically created by the component library at application startup, so further instances of the TApplication component should not be created.

Properties	Methods	Events
AutoFocus	CreateDatabase	OnError
IdleTimeout	CreateForm	OnIdle
InertiaScrollDuration	CreateRequestHandler	
InertiaScrollStyle	Run	
InertiaScrollThreshhold		
IsAndroid		
IsIOS		
IsWindowsPhone		
LoadProgress		
MainForm		
MainRequestHandler		
Request		
Surface		
Title		
TouchScrollThreshhold		
Version		
Viewport		

TApplication.AutoFocus Property

```
property AutoFocus: Boolean
```

Available In: Visual Client Applications

Specifies whether an application should automatically set focus to focusable controls when showing forms, as well as restoring focus to the last-focused control when hiding forms. This property is set to True, by default, with desktop browsers, and to False, by default, with mobile browsers on Android and iOS.

TApplication.IdleTimeout Property

```
property IdleTimeout: Integer
```

Available In: Visual Client Applications

Specifies the time, in seconds, that a visual client application should wait on user input (keypresses, mouse clicks, or touches) before triggering the OnIdle event. This is useful for functionality such as making sure that any authentication information cached for the current user is discarded after a certain period of inactivity, thus forcing the user to login again when interaction with the application is resumed. The default value is 300 seconds, or 5 minutes.

> **Note**
> Mouse movement alone is not enough to reset the idle timeout. The user must specifically press a key or mouse button, or touch the surface of the screen.

TApplication.InertiaScrollDuration Property

```
property InertiaScrollDuration: Integer
```

Available In: Visual Client Applications

Specifies the total amount of time, in milliseconds, that inertia scrolling will take place after the user has lifted their finger from the touch surface. The default value is 1950 milliseconds.

TApplication.InertiaScrollStyle Property

```
property InertiaScrollStyle: TAnimationStyle
```

Available In: Visual Client Applications

Specifies the type of animation to use for performing inertia scrolling after the user has lifted their finger from the touch surface. The animation type determines how the scroll velocity is adjusted over the entire InertiaScrollDuration. The default value is asQuadEaseOut.

TApplication.InertiaScrollThreshhold Property

```
property InertiaScrollThreshhold: Integer
```

Available In: Visual Client Applications

Specifies the finger movement velocity, in pixels per second, required on the touch surface before a touch movement will result in inertia scrolling after the user has lifted their finger from the touch surface. The default value is 10 pixels per second.

TApplication.IsAndroid Property

```
property IsAndroid: Boolean
```

Available In: Visual Client Applications

Indicates whether the platform running the application is the Android platform.

TApplication.IsIOS Property

```
property IsIOS: Boolean
```

Available In: Visual Client Applications

Indicates whether the platform running the application is the IOS platform.

TApplication.IsWindowsPhone Property

```
property IsWindowsPhone: Boolean
```

Available In: Visual Client Applications

Indicates whether the platform running the application is the Windows Phone platform.

TApplication.LoadProgress Property

```
property LoadProgress: Boolean
```

Available In: Visual Client Applications

Specifies whether load progress should be shown during the initialization and loading of an application. This is useful for applications that have many auto-create forms that may take some time to create at application startup.

> **Note**
> This property is automatically set during application initialization from settings specified in the Project Options dialog.

TApplication.MainForm Property

```
property MainForm: TFormControl
```

Available In: Visual Client Applications

Indicates the auto-created form that is designated as the main form for the project. This property can be modified via the **Startup** page of the Project Options dialog. This property cannot be modified at run-time.

TApplication.MainRequestHandler Property

```
property MainRequestHandler: TRequestHandler
```

Available In: Server Applications

Indicates the auto-created request handler that is designated as the main request handler for the project. This property can be modified via the **Startup** page of the Project Options dialog. This property cannot be modified at run-time.

TApplication.Request Property

```
property Request: TWebServerRequest
```

Available In: Server Applications

Contains a reference to the incoming web server request.

TApplication.Surface Property

```
property Surface: TSurface
```

Available In: Visual Client Applications

Contains a reference to the application's surface, which acts as the parent control to all forms and controls in a visual client application.

TApplication.Title Property

```
property Title: String

property Title: String
```

Available In: Visual Client and Server Applications

Indicates the title of the application. For visual client applications, the title appears in the web browser for the application's tab or page. For server applications, the title can be used for error messages and other identification purposes. This property can be modified at design-time via the **General** page of the Project Options dialog, and can also be modified at run-time in visual client applications to specify a different title.

Note
This property is automatically set during application initialization from settings specified in the Project Options dialog.

TApplication.TouchScrollThreshhold Property

```
property TouchScrollThreshhold: Integer
```

Available In: Visual Client Applications

Specifies the amount of finger movement, in pixels, required on the touch surface before a touch movement is considered a scroll movement. The default value is 4 pixels.

TApplication.Version Property

```
property Version: String

property Version: String
```

Available In: Visual Client and Server Applications

Indicates the version of the application, which can be used for error messages and other identification purposes. This property can be modified at design-time via the **General** page of the Project Options dialog. This property cannot be modified at run-time.

Note
This property is automatically set during application initialization from settings specified in the Project Options dialog.

TApplication.Viewport Property

```
property Viewport: TViewport
```

Available In: Visual Client Applications

Contains a reference to the application's viewport, which provides information about the dimensions of the browser window or container and allows the developer to specify how the applicaton surface should be scrolled within the bounds of the browser window.

TApplication.CreateDatabase Method

```
procedure CreateDatabase(ADatabaseClass: TDatabaseClass)
```

Available In: Visual Client and Server Applications

This method is called during application startup to create any databases marked for auto-creation, and should not be called manually. The list of auto-created designer instances can be modified at design-time via the **Startup** page of the Project Options dialog.

TApplication.CreateForm Method

```
procedure CreateForm(AFormClass: TFormControlClass)
```

Available In: Visual Client Applications

This method is called during application startup to create any forms marked for auto-creation, and should not be called manually. The list of auto-created designer instances can be modified at design-time via the **Startup** page of the Project Options dialog.

TApplication.CreateRequestHandler Method

```
procedure CreateRequestHandler(ARequestHandlerClass:
      TRequestHandlerClass)
```

Available In: Server Applications

This method is called during application startup to create any request handlers marked for auto-creation, and should not be called manually. The list of auto-created designer instances can be modified at design-time via the **Startup** page of the Project Options dialog.

TApplication.Run Method

```
procedure Run(const AMainFormName: String='')

procedure Run(const AMainRequestHandlerName: String='')
```

Available In: Visual Client and Server Applications

This method is automatically called by the framework during application startup, and should not be called manually.

TApplication.OnError Event

```
property OnError: TErrorEvent

property OnError: TErrorEvent
```

Available In: Visual Client and Server Applications

This event is triggered when an exception occurs in an application and is not handled by any local try..exception blocks in the code.

TApplication.OnIdle Event

```
property OnIdle: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when an application's IdleTimeout property has been exceeded.

Note
This event is only fired once per period of inactivity, and is not fired again until mouse/touch/keyboard activity or a modification to the IdleTimeout property causes it to be reset.

10.9 TAudio Component

Unit: WebMedia

Inherits From TMediaControl

Available In: Visual Client Applications

The TAudio control encapsulates the HTML5 audio support available in web browsers. With the TAudio control, you can handle most aspects of audio loading and playback.

> **Note**
> This control is a visual control because it can, optionally, display a UI for controlling playback, volume, etc.

Properties	Methods	Events
AutoPlay		OnAbort
CurrentTime		OnAnimationComplete
Cursor		OnAnimationsComplete
DataColumn		OnCanPlay
DataSet		OnCanPlayThrough
DefaultPlaybackRate		OnCaptureEnd
Duration		OnCaptureStart
Ended		OnCapturing
Hint		OnClick
Loop		OnContextMenu
Muted		OnDblClick
NetworkState		OnDurationChange
Opacity		OnEmptied
Paused		OnEnded
PlaybackRate		OnError
Preload		OnHide
ReadyState		OnLoadedData
Seeking		OnLoadedMetadata
ShowControls		OnLoadStart
SourceURL		OnMouseDown
Volume		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp

		OnMove
		OnPause
		OnPlay
		OnPlaying
		OnProgress
		OnRateChange
		OnSeeked
		OnSeeking
		OnShow
		OnSize
		OnStalled
		OnSuspend
		OnTimeUpdate
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart
		OnVolumeChange
		OnWaiting

TAudio.AutoPlay Property

```
property AutoPlay: Boolean
```

Available In: Visual Client Applications

Specifies that the audio should begin playing as soon as enough data has been loaded to allow playback. The default value is False.

TAudio.CurrentTime Property

```
property CurrentTime: Double
```

Available In: Visual Client Applications

Indicates the current playback time, in seconds. Setting this property to a new value will cause the audio to skip to the specified time.

TAudio.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TAudio.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TAudio.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

TAudio.DefaultPlaybackRate Property

```
property DefaultPlaybackRate: Double
```

Available In: Visual Client Applications

Specifies the default playback rate, with 1 being normal playback, less than 1 being slower playback, and greater than 1 being faster playback. The default value is 1.

Note
The volume will normally be automatically muted when playing audio faster or slower than the normal playback rate.

TAudio.Duration Property

```
property Duration: Double
```

Available In: Visual Client Applications

Indicates the length of the audio in seconds. Add an event handler for the OnDurationChange event to detect when the duration has been determined for the current audio being loaded/played. If the duration has not been determined, this property will return 0.

TAudio.Ended Property

```
property Ended: Boolean
```

Available In: Visual Client Applications

Indicates that the end of the audio has been reached.

TAudio.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TAudio.Loop Property

```
property Loop: Boolean
```

Available In: Visual Client Applications

Specifies that the audio playback should automatically restart at the beginning once the end has been reached. The default value is False.

TAudio.Muted Property

```
property Muted: Boolean
```

Available In: Visual Client Applications

Specifies that the playback volume should be muted. The default valuse is False.

TAudio.NetworkState Property

```
property NetworkState: TMediaNetworkState
```

Available In: Visual Client Applications

Indicates the network state of the audio loading/playback.

TAudio.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TAudio.Paused Property

```
property Paused: Boolean
```

Available In: Visual Client Applications

Indicates that audio playback is paused, either by the user pausing the audio via the user interface when the ShowControls property is True, or by the application calling the Pause method. The default value is False.

TAudio.PlaybackRate Property

```
property PlaybackRate: Double
```

Available In: Visual Client Applications

Specifies the playback rate, with 1 being normal playback, less than 1 being slower playback, and greater than 1 being faster playback. The default value is 1.

Note
The volume will normally be automatically muted when playing audio faster or slower than the normal playback rate.

TAudio.Preload Property

```
property Preload: TMediaPreload
```

Available In: Visual Client Applications

Specifies how much of the current audio data should be loaded before playback begins.

TAudio.ReadyState Property

```
property ReadyState: TMediaReadyState
```

Available In: Visual Client Applications

Indicates whether the audio is ready for playback, and if so, a general description of what audio data has been loaded.

TAudio.Seeking Property

```
property Seeking: Boolean
```

Available In: Visual Client Applications

Indicates that audio is switching to a new playback location, either by the user changing the playback location in the audio via the user interface when the ShowControls property is True, or by the application setting the CurrentTime property.

TAudio.ShowControls Property

```
property ShowControls: Boolean
```

Available In: Visual Client Applications

Specifies whether the control should show the native user interface for the audio being played.

TAudio.SourceURL Property

```
property SourceURL: String
```

Available In: Visual Client Applications

Specifies the URL of the audio to be loaded into the control. Whenever this property is changed, the existing audio is cleared and the new audio will start downloading from the web server. Please review the events available for this control in order to get more information on detecting and handling the loading/playback of the audio.

TAudio.Volume Property

```
property Volume: Integer
```

Available In: Visual Client Applications

Specifies the playback volume of the audio. The volume can be set between 0 and 100.

TAudio.OnAbort Event

```
property OnAbort: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media control has stopped loading data for the current media. This is normally caused by the user requesting such an action via the user interface when the ShowControls property is True.

TAudio.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TAudio.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TAudio.OnCanPlay Event

```
property OnCanPlay: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media control has loaded enough data to begin playback. However, additional data loading may be required as playback continues.

TAudio.OnCanPlayThrough Event

```
property OnCanPlayThrough: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media control has loaded enough data to begin playback and (probably) play the media until the end without needing to load any additional data.

TAudio.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TAudio.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TAudio.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TAudio.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TAudio.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TAudio.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TAudio.OnDurationChange Event

```
property OnDurationChange: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the duration of the media changes, which normally occurs when loading new media into the control by modifying the SourceURL property.

TAudio.OnEmptied Event

```
property OnEmptied: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever an error or abort has caused the NetworkState property to revert to the mnsEmpty state.

TAudio.OnEnded Event

```
property OnEnded: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever playback has stopped because the end of the media has been reached.

TAudio.OnError Event

```
property OnError: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever an error has prevented the media from being loaded properly.

TAudio.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TAudio.OnLoadedData Event

```
property OnLoadedData: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media control has loaded enough data for the current playback location.

TAudio.OnLoadedMetadata Event

```
property OnLoadedMetadata: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media control has loaded the metadata, including the duration and dimensions, for the current media.

TAudio.OnLoadStart Event

```
property OnLoadStart: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media control starts loading data for the current media.

TAudio.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TAudio.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TAudio.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TAudio.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TAudio.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TAudio.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TAudio.OnPause Event

```
property OnPause: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media playback is paused.

TAudio.OnPlay Event

```
property OnPlay: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media playback is started/resumed.

TAudio.OnPlaying Event

```
property OnPlaying: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever media playback has actually started.

Note
This event is slightly different from the OnPlay event, which only indicates that the user or application **requested** playback to start/resume. This event may be triggered multiple times during playback, especially if playback needs to stop in order to allow more media data to be loaded, which can be the case with slower network connections.

TAudio.OnProgress Event

```
property OnProgress: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the current media is being loaded.

Note
This event is typically fired several times per second in most web browsers, so be very careful about how time-consuming any event handlers are for this event.

TAudio.OnRateChange Event

```
property OnRateChange: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the playback rate of the media control has changed for the current media. This is caused by the user requesting such an action via the user interface when the ShowControls property is True, or when the application modifies the PlaybackRate property.

TAudio.OnSeeked Event

```
property OnSeeked: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the Seeking property reverts to False.

TAudio.OnSeeking Event

```
property OnSeeking: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the playback location of the media control has changed for the current media. This is caused by the user requesting such an action via the user interface when the ShowControls property is True, or when the application modifies the CurrentTime property.

TAudio.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TAudio.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TAudio.OnStalled Event

```
property OnStalled: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media control is trying to load data for the current media, but no data is arriving over the network.

TAudio.OnSuspend Event

```
property OnSuspend: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media control has loaded enough data to enable playback, and has stopped loading more data.

TAudio.OnTimeUpdate Event

```
property OnTimeUpdate: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the CurrentTime property changes.

Note
This event can be fired as many as 60 times per second in some web browsers, so be very careful about how time-consuming any event handlers are for this event.

TAudio.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TAudio.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TAudio.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TAudio.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

TAudio.OnVolumeChange Event

```
property OnVolumeChange: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the audio volume of the media control has changed for the current media. This is caused by the user requesting such an action via the user interface when the ShowControls property is True, or when the application modifies the Volume property.

TAudio.OnWaiting Event

```
property OnWaiting: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media control cannot start/resume playback because more data needs to be loaded for the current media.

10.10 TAudioElement Component

Unit: WebUI

Inherits From TMediaElement

Available In: Visual Client Applications

The TAudioElement class is the element class for audio UI elements, and contains all of the audio playback functionality in the form of public methods and properties/events that control classes can use to create audio controls.

> **Note**
> This element does not provide support for audio playback at design-time, and the applicable playback methods and properties are all stubs.

Properties	Methods	Events

10.11 TAutoSize Component

Unit: WebUI

Inherits From TElementAttribute

Available In: Visual Client Applications

The TAutoSize class represents the auto-sizing attributes to use for the content of a UI element or control. The height, width, or both can be set as auto-sized. The content of a UI element or control that is used to determine the height and/or width can be text or HTML content, as well as the space consumed by child UI elements or controls.

> **Note**
> Not all controls support auto-sizing.

Properties	Methods	Events
Height	SetToDefault	
Width		

TAutoSize.Height Property

```
property Height: Boolean
```

Available In: Visual Client Applications

Specifies that the height of the UI element or control should be automatically set based upon the height of its content and/or the height of any child UI elements or controls.

TAutoSize.Width Property

```
property Width: Boolean
```

Available In: Visual Client Applications

Specifies that the width of the UI element or control should be automatically set based upon the width of its content and/or the width of any child UI elements or controls.

TAutoSize.SetToDefault Method

```
procedure SetToDefault
```

Available In: Visual Client Applications

Use this method to reset all auto-size properties to their default values.

10.12 TBackground Component

Unit: WebUI

Inherits From TElementAttribute

Available In: Visual Client Applications

The TBackground class represents the background of a UI element or control. Backgrounds can be solid colors (including transparent) or gradients, and can have tiled and non-tiled background images.

Properties	Methods	Events
Clip	SetToDefault	
Fill		
Image		
Origin		

TBackground.Clip Property

```
property Clip: TBackgroundOrientationType
```

Available In: Visual Client Applications

Specifies how the background should be clipped within the UI element or control.

TBackground.Fill Property

```
property Fill: TFill
```

Available In: Visual Client Applications

Specifies the background fill.

TBackground.Image Property

```
property Image: TBackgroundImage
```

Available In: Visual Client Applications

Specifies the background image.

TBackground.Origin Property

```
property Origin: TBackgroundOrientationType
```

Available In: Visual Client Applications

Specifies the origin of the background within the UI element or control.

TBackground.SetToDefault Method

```
procedure SetToDefault
```

Available In: Visual Client Applications

Use this method to reset the background's properties to their default values.

10.13 TBackgroundImage Component

Unit: WebUI

Inherits From TElementAttribute

Available In: Visual Client Applications

The TBackgroundImage class represents the background image of a UI element or control. The background image is specified using the Name property and/or the Data (for base64-encoded, data-URI inline images).

> **Note**
> Any background images specified at design-time are automatically embedded in form and control interface files, are emitted as part of the application during compilation, and are automatically loaded during application initialization.

Properties	Methods	Events
Data	BeginAnimation	
Height	CancelAnimation	
Left	SetToDefault	
Name		
PositionType		
RepeatStyle		
SizeType		
Top		
Width		

TBackgroundImage.Data Property

```
property Data: String
```

Available In: Visual Client Applications

Specifies the image as a base64-encoded, data-URI inline image. The image data should **not** include any data-URI prefixes. The interface manager automatically adds the proper prefixes when applying the background to the applicable UI element or control.

Note
If the Name property **and** the Data property are both assigned non-blank values, then the Name property is effectively ignored. Also, if the Name property is assigned a new value, the Data property will automatically be cleared.

TBackgroundImage.Height Property

```
property Height: Integer
```

Available In: Visual Client Applications

Specifies the height of the background image. If the actual image height of the background image is different than this value, then the background image height is stretched/contracted accordingly.

> **Note**
> This property is only valid when the SizeType is stSpecified.

TBackgroundImage.Left Property

```
property Left: Integer
```

Available In: Visual Client Applications

Specifies the left position of the background image.

Note
This property is only valid when the PositionType is ptSpecified.

TBackgroundImage.Name Property

```
property Name: String
```

Available In: Visual Client Applications

Specifies the local image name, at design-time, or the URL of an image resource, at run-time.

Note
If the Name property **and** the Data property are both assigned non-blank values, then the Name property is effectively ignored. Also, if the Name property is assigned a new value, the Data property will automatically be cleared.

TBackgroundImage.PositionType Property

```
property PositionType: TBackgroundImagePositionType
```

Available In: Visual Client Applications

Specifies how the background image should be positioned within the UI element or control.

Note
The bounding rectangle used for determining the positioning is based upon the TBackground Origin and Clip properties.

TBackgroundImage.RepeatStyle Property

```
property RepeatStyle: TBackgroundImageRepeatStyle
```

Available In: Visual Client Applications

Specifies how the background image should be tiled within the UI element or control.

Note
The bounding rectangle used for determining the positioning is based upon the TBackground Origin and Clip properties.

TBackgroundImage.SizeType Property

```
property SizeType: TBackgroundImageSizeType
```

Available In: Visual Client Applications

Specifies how the background image should be sized within the UI element or control.

Note
The bounding rectangle used for determining the positioning is based upon the TBackground Origin and Clip properties.

TBackgroundImage.Top Property

```
property Top: Integer
```

Available In: Visual Client Applications

Specifies the top position of the background image.

Note
This property is only valid when the PositionType is ptSpecified.

TBackgroundImage.Width Property

```
property Width: Integer
```

Available In: Visual Client Applications

Specifies the width of the background image. If the actual image width of the background image is different than this value, then the background image width is stretched/contracted accordingly.

> **Note**
> This property is only valid when the SizeType is stSpecified.

TBackgroundImage.BeginAnimation Method

```
procedure BeginAnimation(ADirection:
      TBackgroundImageAnimateDirection)
```

Available In: Visual Client Applications

Use this method to begin animating the collection of animation frames in a background image.

TBackgroundImage.CancelAnimation Method

```
procedure CancelAnimation
```

Available In: Visual Client Applications

Use this method to stop animating the collection of animation frames in a background image.

TBackgroundImage.SetToDefault Method

```
procedure SetToDefault
```

Available In: Visual Client Applications

Use this method to reset the background image's properties to their default values.

10.14 TBalloonLabel Component

Unit: WebLabels

Inherits From TBalloonLabelControl

Available In: Visual Client Applications

The TBalloonLabel component represents a label control that looks like a word balloon, with customizations such as the orientation of the balloon tail and the ability to specify an icon along with the text.

Properties	Methods	Events
AutoHideTime		OnAnimationComplete
Background		OnAnimationsComplete
Caption		OnCaptureEnd
Corners		OnCaptureStart
Cursor		OnCapturing
DataColumn		OnClick
DataSet		OnContextMenu
Font		OnDblClick
Format		OnHide
Hint		OnMouseDown
Icon		OnMouseEnter
Opacity		OnMouseLeave
Orientation		OnMouseMove
Padding		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TBalloonLabel.AutoHideTime Property

```
property AutoHideTime: Integer
```

Available In: Visual Client Applications

Specifies the number of milliseconds that the balloon label control should be shown before automatically hiding itself. The default value is 0 milliseconds, which prevents the control from automatically hiding itself.

TBalloonLabel.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

TBalloonLabel.Caption Property

```
property Caption: TCaption
```

Available In: Visual Client Applications

Specifies the text to be shown in the balloon label control. The text can contain line feeds. The default value is ''.

TBalloonLabel.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TBalloonLabel.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TBalloonLabel.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TBalloonLabel.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

TBalloonLabel.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TBalloonLabel.Format Property

```
property Format: TFormat
```

Available In: Visual Client Applications

Specifies the content formatting to use for the control's Caption.

TBalloonLabel.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TBalloonLabel.Icon Property

```
property Icon: TIconProperties
```

Available In: Visual Client Applications

Specifies the properties of the icon used with the control.

TBalloonLabel.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TBalloonLabel.Orientation Property

```
property Orientation: TBalloonOrientation
```

Available In: Visual Client Applications

Specifies the orientation of the tail for the balloon label.

TBalloonLabel.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

TBalloonLabel.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TBalloonLabel.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TBalloonLabel.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TBalloonLabel.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TBalloonLabel.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TBalloonLabel.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TBalloonLabel.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TBalloonLabel.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TBalloonLabel.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TBalloonLabel.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TBalloonLabel.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TBalloonLabel.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TBalloonLabel.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TBalloonLabel.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TBalloonLabel.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TBalloonLabel.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TBalloonLabel.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TBalloonLabel.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TBalloonLabel.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TBalloonLabel.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TBalloonLabel.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.15 TBalloonLabelControl Component

Unit: WebLabels

Inherits From TBindableColumnControl

Available In: Visual Client Applications

The TBalloonLabelControl control is the base class for balloon label controls, and contains all of the label functionality in the form of public methods and protected properties/events that descendant classes can use to create customized balloon label controls.

Properties	Methods	Events

10.16 TBasicPanel Component

Unit: WebCtnrs

Inherits From TBasicPanelControl

Available In: Visual Client Applications

The TBasicPanel component represents a basic panel control that cannot be scrolled.

Properties	Methods	Events
ActivateOnClick		OnAnimationComplete
AutoSize		OnAnimationsComplete
Background		OnCaptureEnd
Border		OnCaptureStart
Corners		OnCapturing
Cursor		OnClick
Hint		OnContextMenu
InsetShadow		OnDblClick
Opacity		OnHide
OutsetShadow		OnKeyDown
Padding		OnKeyPress
TabOrder		OnKeyUp
TabStop		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TBasicPanel.ActivateOnClick Property

```
property ActivateOnClick: Boolean
```

Available In: Visual Client Applications

Specifies whether the control should automatically be brought to the front when it, or any child controls, are clicked.

TBasicPanel.AutoSize Property

```
property AutoSize: TAutoSize
```

Available In: Visual Client Applications

Specifies how (if at all) the control should automatically be sized based upon the child controls placed in the panel.

TBasicPanel.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

TBasicPanel.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TBasicPanel.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TBasicPanel.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TBasicPanel.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TBasicPanel.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Available In: Visual Client Applications

Specifies the inset shadow for the control.

TBasicPanel.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TBasicPanel.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Available In: Visual Client Applications

Specifies the outset shadow for the control.

TBasicPanel.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

TBasicPanel.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TBasicPanel.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TBasicPanel.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TBasicPanel.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TBasicPanel.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TBasicPanel.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TBasicPanel.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TBasicPanel.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TBasicPanel.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TBasicPanel.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TBasicPanel.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TBasicPanel.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and the user presses a key or key combination.

TBasicPanel.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and presses/releases a key or key combination.

TBasicPanel.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and the user releases a key or key combination.

TBasicPanel.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TBasicPanel.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TBasicPanel.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TBasicPanel.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TBasicPanel.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TBasicPanel.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TBasicPanel.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TBasicPanel.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TBasicPanel.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TBasicPanel.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TBasicPanel.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TBasicPanel.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.17 TBasicPanelControl Component

Unit: WebCtnrs

Inherits From TControl

Available In: Visual Client Applications

The TBasicPanelControl control is the base class for basic panel controls, and contains all of the panel functionality in the form of public methods and protected properties/events that descendant classes can use to create customized panel controls.

Properties	Methods	Events

10.18 TBindableColumnControl Component

Unit: WebCtrls

Inherits From TBindableControl

Available In: Visual Client Applications

The TBindableColumnControl control is the base class for controls that bind to dataset columns, and contains all of the dataset column binding functionality in the form of public methods and protected properties/events that descendant classes can use to create customized controls that bind to dataset columns.

Properties	Methods	Events

10.19 TBindableControl Component

Unit: WebCtrls

Inherits From TControl

Available In: Visual Client Applications

The TBindableControl control is the base class for controls that bind to datasets. It contains all of the dataset binding functionality in the form of public methods and protected properties/events that descendant classes can use to create customized controls that bind to datasets.

Properties	Methods	Events

10.20 TBodyElement Component

Unit: WebUI

Inherits From TElement

Available In: Visual Client Applications

The TBodyElement class is the element class used as the root element by the interface manager at run-time. It wraps the browser document's body element, and a visual application's Surface uses this element as its base element.

Properties	Methods	Events

10.21 TBorder Component

Unit: WebUI

Inherits From TElementAttribute

Available In: Visual Client Applications

The TBorder class represents the border for a UI element or control.

Properties	Methods	Events
Bottom	SetToDefault	
Left		
Right		
Top		

TBorder.Bottom Property

```
property Bottom: TBorderSide
```

Available In: Visual Client Applications

Specifies the bottom border side.

TBorder.Left Property

```
property Left: TBorderSide
```

Available In: Visual Client Applications

Specifies the left border side.

TBorder.Right Property

```
property Right: TBorderSide
```

Available In: Visual Client Applications

Specifies the right border side.

TBorder.Top Property

```
property Top: TBorderSide
```

Available In: Visual Client Applications

Specifies the top border side.

TBorder.SetToDefault Method

```
procedure SetToDefault
```

Available In: Visual Client Applications

Use this method to reset the border's properties to their default values.

10.22 TBorderSide Component

Unit: WebUI

Inherits From TElementAttribute

Available In: Visual Client Applications

The TBorderSide class represents one side of the TBorder for a UI element or control.

Properties	Methods	Events
Color	SetToDefault	
Offset		
Style		
Visible		
Width		

TBorderSide.Color Property

```
property Color: TColor
```

Available In: Visual Client Applications

Specifies the color of the border side.

TBorderSide.Offset Property

```
property Offset: Integer
```

Available In: Visual Client Applications

Indicates the actual offset, in pixels, of the border. At run-time in a browser, a UI element or control's border side Width property is only taken into account when the border side is visible.

TBorderSide.Style Property

```
property Style: TBorderStyle
```

Available In: Visual Client Applications

Specifies the style of the border side.

TBorderSide.Visible Property

```
property Visible: Boolean
```

Available In: Visual Client Applications

Specifies whether the border side is visible.

TBorderSide.Width Property

```
property Width: Integer
```

Available In: Visual Client Applications

Specifies the width, in pixels, of the border side.

TBorderSide.SetToDefault Method

```
procedure SetToDefault
```

Available In: Visual Client Applications

Use this method to reset the border side's properties to their default values.

10.23 TBoundingAttribute Component

Unit: WebUI

Inherits From TElementAttribute

Available In: Visual Client Applications

The TBoundingAttribute class represents a series of left, top, right, and bottom bounding values, in pixels, for a UI element or control. It is the base class for the TMargins and TPadding classes.

Properties	Methods	Events
Bottom	SetToDefault	
Left		
Right		
Top		

TBoundingAttribute.Bottom Property

```
property Bottom: Integer
```

Available In: Visual Client Applications

Specifies the bottom bounding value.

TBoundingAttribute.Left Property

```
property Left: Integer
```

Available In: Visual Client Applications

Specifies the left bounding value.

TBoundingAttribute.Right Property

```
property Right: Integer
```

Available In: Visual Client Applications

Specifies the right bounding value.

TBoundingAttribute.Top Property

```
property Top: Integer
```

Available In: Visual Client Applications

Specifies the top bounding value.

TBoundingAttribute.SetToDefault Method

```
procedure SetToDefault
```

Available In: Visual Client Applications

Use this method to reset the bounding attribute properties to their default values.

10.24 TBrowser Component

Unit: WebBrwsr

Inherits From TWebControl

Available In: Visual Client Applications

The TBrowser component represents an HTML document container control, and can be used as the output destination for HTML Forms via the THTMLForm Output property.

Properties	Methods	Events
Background	Print	OnAnimationComplete
Border		OnAnimationsComplete
Corners		OnHide
Cursor		OnLoad
DataColumn		OnMove
DataSet		OnShow
Document		OnSize
DocumentText		OnUnload
Hint		
InsetShadow		
Loaded		
Opacity		
OutsetShadow		
Padding		
Scrolling		
URL		

TBrowser.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background for the control.

TBrowser.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TBrowser.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TBrowser.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TBrowser.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TBrowser.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

TBrowser.Document Property

```
property Document: THTMLDocument
```

Available In: Visual Client Applications

Returns the DOM (Document Object Model) document instance of the currently-loaded HTML document. If the URL property has been specified, then this property will return the document instance once the OnLoad event has been triggered and the Loaded property is True.

Note
Accessing the DOM document instance allows you to manipulate the children of the DOM document instance in code instead of having to use HTML strings, which is the case when using the DocumentText property. However, this access is subject to same-origin security constraints, and will be denied if the contents of the TBrowser instance were loaded from a different origin.

TBrowser.DocumentText Property

```
property DocumentText: String
```

Available In: Visual Client Applications

Returns the currently-loaded HTML document in the control as a string. If the URL property has been specified, then this property will return the document contents once the OnLoad event has been triggered and the Loaded property is True.

> **Note**
> You can also assign a valid HTML string to this property, in which case the URL property is automatically cleared.

TBrowser.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TBrowser.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Available In: Visual Client Applications

Specifies the inset shadow for the control.

TBrowser.Loaded Property

```
property Loaded: Boolean
```

Available In: Visual Client Applications

Indicates whether the HTML document specified by the URL property has been loaded.

An event handler can be attached to the OnLoad event to execute code when the document is loaded.

TBrowser.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TBrowser.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Available In: Visual Client Applications

Specifies the outset shadow for the control.

TBrowser.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

TBrowser.Scrolling Property

```
property Scrolling: Boolean
```

Available In: Visual Client Applications

Specifies whether scrolling should be enabled for the control.

Note
This property is required because certain browsers require a special way of specifying whether scrollbars should be shown for embedded browser controls.

TBrowser.URL Property

```
property URL: String
```

Available In: Visual Client Applications

Specifies the URL for the HTML document. Whenever the URL property changes, the OnUnload event is triggered immediately. The OnLoad event is triggered once the document has been loaded.

TBrowser.Print Method

```
procedure Print
```

Available In: Visual Client Applications

Use this method to print the currently-loaded HTML document in the control. If no HTML document is loaded, then this method does nothing.

TBrowser.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TBrowser.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TBrowser.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TBrowser.OnLoad Event

```
property OnLoad: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the HTML document specified by the URL property has been completely loaded.

TBrowser.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TBrowser.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TBrowser.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TBrowser.OnUnload Event

```
property OnUnload: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the currently-loaded HTML document specified by the URL or DocumentText property has been unloaded.

10.25 TButton Component

Unit: WebBtns

Inherits From TButtonControl

Available In: Visual Client Applications

The TButton component represents a button control. A button control allows the user to initiate a specific action by using a mouse click or by pushing the spacebar or enter key.

Properties	Methods	Events
AutoWidth		OnAnimationComplete
Caption		OnAnimationsComplete
Corners		OnCaptureEnd
Cursor		OnCaptureStart
Enabled		OnCapturing
Font		OnClick
Hint		OnContextMenu
Icon		OnEnter
TabOrder		OnExit
TabStop		OnHide
		OnKeyDown
		OnKeyPress
		OnKeyUp
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TButton.AutoWidth Property

```
property AutoWidth: Boolean
```

Available In: Visual Client Applications

Specifies whether the width of the button should be automatically set based upon the Caption, Icon, and Font properties.

TButton.Caption Property

```
property Caption: String
```

Available In: Visual Client Applications

Specifies the textual caption to display on the button control. The default value is ''.

TButton.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TButton.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TButton.Enabled Property

```
property Enabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TButton.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the font to use for the caption on the button control.

TButton.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TButton.Icon Property

```
property Icon: TIconProperties
```

Available In: Visual Client Applications

Specifies the properties of the icon used with the control.

TButton.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TButton.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TButton.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TButton.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TButton.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TButton.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TButton.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TButton.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TButton.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TButton.OnEnter Event

```
property OnEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control obtains input focus.

TButton.OnExit Event

```
property OnExit: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control loses input focus.

TButton.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TButton.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses a key or key combination.

TButton.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TButton.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user releases a key or key combination.

TButton.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TButton.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TButton.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TButton.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TButton.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TButton.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TButton.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TButton.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TButton.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TButton.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TButton.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TButton.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.26 TButtonComboBox Component

Unit: WebEdits

Inherits From TButtonComboControl

Available In: Visual Client Applications

The TButtonComboBox component represents a button combo box control. A button combo box is a combo control that allows the user to select an input value from a drop-down list of values by using a mouse click or by pushing the spacebar or enter key.

> **Note**
> This type of control does not allow for direct editing of the input value, and is ideal for touch environments where you may not want a soft keyboard to appear when such a control gains focus.

Properties	Methods	Events
Alignment		OnAnimationComplete
AutoDropDown		OnAnimationsComplete
AutoItemHeight		OnCaptureEnd
Cursor		OnCaptureStart
DataColumn		OnCapturing
DataSet		OnChange
Direction		OnClick
DropDownItemCount		OnContextMenu
DropDownPosition		OnDropDownHide
DropDownVisible		OnDropDownShow
Enabled		OnEnter
Font		OnExit
Hint		OnHide
ItemHeight		OnKeyDown
ItemIndex		OnKeyPress
Items		OnKeyUp
KeyPressInterval		OnMouseDown
ReadOnly		OnMouseEnter
Sorted		OnMouseLeave
TabOrder		OnMouseMove
TabStop		OnMouseUp
Text		OnMove
		OnShow

		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TButtonComboBox.Alignment Property

```
property Alignment: TContentAlignment
```

Available In: Visual Client Applications

Specifies the alignment of the input value for the control.

TButtonComboBox.AutoDropDown Property

```
property AutoDropDown: Boolean
```

Available In: Visual Client Applications

Specifies that the drop-down list of Items should automatically be shown when the user starts typing. The default value is False.

TButtonComboBox.AutoItemHeight Property

```
property AutoItemHeight: Boolean
```

Available In: Visual Client Applications

Specifies that the displayed height of the drop-down items will automatically be set based upon the Font property settings. The default value is True.

TButtonComboBox.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TButtonComboBox.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TButtonComboBox.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

TButtonComboBox.Direction Property

```
property Direction: TContentDirection
```

Available In: Visual Client Applications

Specifies the direction in which the caption is displayed.

TButtonComboBox.DropDownItemCount Property

```
property DropDownItemCount: Integer
```

Available In: Visual Client Applications

Specifies the number of visible items to display in the drop-down list of Items.

TButtonComboBox.DropDownPosition Property

```
property DropDownPosition: TDropDownPosition
```

Available In: Visual Client Applications

Specifies where the drop-down list will be positioned.

TButtonComboBox.DropDownVisible Property

```
property DropDownVisible: Boolean
```

Available In: Visual Client Applications

Indicates whether the drop-down list is visible.

TButtonComboBox.Enabled Property

```
property Enabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TButtonComboBox.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TButtonComboBox.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TButtonComboBox.ItemHeight Property

```
property ItemHeight: Integer
```

Available In: Visual Client Applications

Specifies the height, in pixels, of the items displayed in the drop-down list.

TButtonComboBox.ItemIndex Property

```
property ItemIndex: Integer
```

Available In: Visual Client Applications

Specifies the index of the selected item in the drop-down list of Items, or -1 if there is no selected item.

TButtonComboBox.Items Property

```
property Items: TStrings
```

Available In: Visual Client Applications

Specifies the items to use for the drop-down list.

TButtonComboBox.KeyPressInterval Property

```
property KeyPressInterval: Integer
```

Available In: Visual Client Applications

Specifies the interval, in milliseconds, that is used by the control to combine user keystrokes into a search value that is then used for performing a near search on the Items property. Effectively, this means that the user has KeyPressInterval milliseconds in which to hit a key in order for the keystroke to be included as part of a near search. The default value is 300 milliseconds.

For example, if the user hits the "S", "M", and "I" keys within the KeyPressInterval property value, but hits the "T" key outside of the KeyPressInterval property, then the control will perform a near search using the value "SMI", followed by a near search using the value "T".

TButtonComboBox.ReadOnly Property

```
property ReadOnly: Boolean
```

Available In: Visual Client Applications

Specifies whether the control's input value can be modified by the user. The default value is False.

> **Note**
> The input value can always be programmatically modified.

TButtonComboBox.Sorted Property

```
property Sorted: Boolean
```

Available In: Visual Client Applications

Specifies whether the drop-down items will automatically be sorted. The default value is False.

TButtonComboBox.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TButtonComboBox.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TButtonComboBox.Text Property

```
property Text: String
```

Available In: Visual Client Applications

Specifies the control's input value as a string.

TButtonComboBox.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TButtonComboBox.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TButtonComboBox.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TButtonComboBox.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TButtonComboBox.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TButtonComboBox.OnChange Event

```
property OnChange: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TButtonComboBox.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TButtonComboBox.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TButtonComboBox.OnDropDownHide Event

```
property OnDropDownHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the associated drop-down control is hidden.

TButtonComboBox.OnDropDownShow Event

```
property OnDropDownShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the associated drop-down control is shown.

TButtonComboBox.OnEnter Event

```
property OnEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control obtains input focus.

TButtonComboBox.OnExit Event

```
property OnExit: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control loses input focus.

TButtonComboBox.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TButtonComboBox.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses a key or key combination.

TButtonComboBox.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TButtonComboBox.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user releases a key or key combination.

TButtonComboBox.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TButtonComboBox.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TButtonComboBox.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TButtonComboBox.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TButtonComboBox.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TButtonComboBox.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TButtonComboBox.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TButtonComboBox.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TButtonComboBox.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TButtonComboBox.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TButtonComboBox.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TButtonComboBox.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.27 TButtonComboControl Component

Unit: WebEdits

Inherits From TDropDownButtonControl

Available In: Visual Client Applications

The TButtonComboControl control is the base class for button combo controls, and contains all of the button combo functionality in the form of public methods and protected properties/events that descendant classes can use to create customized button combo controls.

Properties	Methods	Events

10.28 TButtonControl Component

Unit: WebBtns

Inherits From TControl

Available In: Visual Client Applications

The TButtonControl control is the base class for button controls, and contains all of the core button functionality in the form of public methods and protected properties/events that descendant classes can use to create customized button controls.

Properties	Methods	Events

10.29 TButtonInputControl Component

Unit: WebEdits

Inherits From TInputControl

Available In: Visual Client Applications

The TButtonInputControl control is the base class for button-style input controls, and contains all of the core input functionality in the form of public methods and protected properties/events that descendant classes can use to create customized button-style input controls.

Properties	Methods	Events

10.30 TCalendar Component

Unit: WebCals

Inherits From TCalendarControl

Available In: Visual Client Applications

The TCalendar component represents a calendar control for selecting a date. The user can change the active period by pressing the page up/page down keys, and can change the calendar View property by pressing the numeric keypad plus and minus keys.

Properties	Methods	Events
Background		OnAnimationComplete
Border		OnAnimationsComplete
Corners		OnCaptureEnd
Cursor		OnCaptureStart
DataColumn		OnCapturing
DataSet		OnChange
DefaultView		OnClick
Enabled		OnContextMenu
Font		OnDblClick
Hint		OnHide
ReadOnly		OnKeyDown
TabOrder		OnKeyPress
TabStop		OnKeyUp
Text		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMouseWheel
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TCalendar.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

TCalendar.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TCalendar.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TCalendar.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TCalendar.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TCalendar.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

TCalendar.DefaultView Property

```
property DefaultView: TCalendarView
```

Available In: Visual Client Applications

Specifies the default view for the calendar control. The default view determines both the initial view shown in the calendar after it is created, as well as the minimum view that the user is permitted to navigate to. The default value is cvMonth.

TCalendar.Enabled Property

```
property Enabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TCalendar.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TCalendar.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TCalendar.ReadOnly Property

```
property ReadOnly: Boolean
```

Available In: Visual Client Applications

Specifies whether the control's input value can be modified by the user. The default value is False.

> **Note**
> The input value can always be programmatically modified.

TCalendar.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TCalendar.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TCalendar.Text Property

```
property Text: String
```

Available In: Visual Client Applications

Specifies the control's input value as a string.

TCalendar.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TCalendar.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TCalendar.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TCalendar.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TCalendar.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TCalendar.OnChange Event

```
property OnChange: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TCalendar.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TCalendar.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TCalendar.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TCalendar.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TCalendar.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses a key or key combination.

TCalendar.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TCalendar.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user releases a key or key combination.

TCalendar.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TCalendar.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TCalendar.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TCalendar.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TCalendar.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TCalendar.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

Available In: Visual Client Applications

This event is triggered whenever the mouse wheel is rotated forward or backward.

TCalendar.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TCalendar.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TCalendar.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TCalendar.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TCalendar.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TCalendar.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TCalendar.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.31 TCalendarControl Component

Unit: WebCals

Inherits From TInputControl

Available In: Visual Client Applications

The TCalendarControl control is the base class for calendar controls, and contains all of the core calendar functionality in the form of public methods and protected properties/events that descendant classes can use to create customized calendar controls.

Properties	Methods	Events
SelectedDate	NextPeriod	
View	PriorPeriod	

TCalendarControl.SelectedDate Property

```
property SelectedDate: DateTime
```

Available In: Visual Client Applications

Specifies the selected date displayed in the calendar.

TCalendarControl.View Property

```
property View: TCalendarView
```

Available In: Visual Client Applications

Specifies the active calendar view.

TCalendarControl.NextPeriod Method

```
procedure NextPeriod
```

Available In: Visual Client Applications

Use this method to navigate to the next period in the calendar control. The next period displayed is determined by the active view.

TCalendarControl.PriorPeriod Method

```
procedure PriorPeriod
```

Available In: Visual Client Applications

Use this method to navigate to the prior period in the calendar control. The prior period displayed is determined by the active view.

10.32 TCanvasElement Component

Unit: WebUI

Inherits From TElement

Available In: Visual Client Applications

The TCanvasElement class is the element class for HTML5 canvas elements, and contains all of the canvas functionality in the form of public methods and properties/events that control classes can use to create painting/drawing controls.

> **Note**
> This element does not provide support for canvas drawing at design-time, and the applicable drawing methods and properties are all stubs.

Properties	Methods	Events
Alpha	Arc	
CompositeOperation	ArcTo	
FillColor	BeginPath	
FillGradient	BezierCurveTo	
FillPattern	ClearRect	
FillStyle	Clip	
LineCapStyle	ClosePath	
LineJoinStyle	ConvertToDataURL	
LineWidth	DrawImage	
MiterLimit	Fill	
ShadowBlur	FillRect	
ShadowColor	FillText	
ShadowOffsetX	IsPointInPath	
ShadowOffsetY	LineTo	
StrokeColor	MeasureText	
StrokeGradient	MoveTo	
StrokePattern	QuadraticCurveTo	
StrokeStyle	Rect	
TextAlign	Rotate	
TextBaseLine	Scale	
	SetTransform	
	Stroke	
	StrokeRect	

	StrokeText	
	Transform	
	Translate	

TCanvasElement.Alpha Property

```
property Alpha: Double
```

Available In: Visual Client Applications

Specifies the global alpha value (transparency) of the canvas. This value is multiplied by the alpha value (transparency) of all drawn pixels. The default is 1.0, meaning that the transparency of the drawn pixels is not modified, and the value must be between 0.0 and 1.0.

TCanvasElement.CompositeOperation Property

```
property CompositeOperation: TCompositeOperation
```

Available In: Visual Client Applications

Specifies how newly-drawn pixels (source) are combined with the existing pixels on the canvas (destination) during drawing operations. The default value is coSourceOver.

TCanvasElement.FillColor Property

```
property FillColor: TColor
```

Available In: Visual Client Applications

Specifies the color to use with fill operations when the FillStyle property is set to dsColor.

TCanvasElement.FillGradient Property

```
property FillGradient: TCanvasGradient
```

Available In: Visual Client Applications

Specifies the gradient to use with fill operations when the FillStyle property is set to dsGradient.

TCanvasElement.FillPattern Property

```
property FillPattern: TCanvasPattern
```

Available In: Visual Client Applications

Specifies the pattern to use with fill operations when the FillStyle property is set to dsPattern.

TCanvasElement.FillStyle Property

```
property FillStyle: TDrawStyle
```

Available In: Visual Client Applications

Specifies how fill operations will draw on the canvas.

TCanvasElement.LineCapStyle Property

```
property LineCapStyle: TLineCapStyle
```

Available In: Visual Client Applications

Specifies how lines should be terminated when drawing wide lines on the canvas. The default value is csButt.

TCanvasElement.LineJoinStyle Property

```
property LineJoinStyle: TLineJoinStyle
```

Available In: Visual Client Applications

Specifies how wide lines should be drawn when they intersect on the canvas. The default value is jsMiter. The MiterLimit property is used to limit the length of a miter join.

TCanvasElement.LineWidth Property

```
property LineWidth: Double
```

Available In: Visual Client Applications

Specifies the line width to use for line drawing (stroking) operations on the canvas.

TCanvasElement.MiterLimit Property

```
property MiterLimit: Double
```

Available In: Visual Client Applications

Specifies the upper limit on miter joins when the LineJoinStyle is set to jsMiter. The default value is 10 (pixels).

TCanvasElement.ShadowBlur Property

```
property ShadowBlur: Double
```

Available In: Visual Client Applications

Specifies how much blur shadows should have. The default value is 0, which produces shadows with a crisp edge.

TCanvasElement.ShadowColor Property

```
property ShadowColor: TColor
```

Available In: Visual Client Applications

Specifies the color of the shadows. The default value is clBlack.

TCanvasElement.ShadowOffsetX Property

```
property ShadowOffsetX: Double
```

Available In: Visual Client Applications

Specifies the horizontal offset of shadows. The default value is 0.

TCanvasElement.ShadowOffsetY Property

```
property ShadowOffsetY: Double
```

Available In: Visual Client Applications

Specifies the vertical offset of shadows. The default value is 0.

TCanvasElement.StrokeColor Property

```
property StrokeColor: TColor
```

Available In: Visual Client Applications

Specifies the color to use with stroke (line-drawing) operations when the StrokeStyle property is set to dsColor.

TCanvasElement.StrokeGradient Property

```
property StrokeGradient: TCanvasGradient
```

Available In: Visual Client Applications

Specifies the gradient to use with stroke (line-drawing) operations when the StrokeStyle property is set to dsGradient.

TCanvasElement.StrokePattern Property

```
property StrokePattern: TCanvasPattern
```

Available In: Visual Client Applications

Specifies the pattern to use with stroke (line-drawing) operations when the StrokeStyle property is set to dsPattern.

TCanvasElement.StrokeStyle Property

```
property StrokeStyle: TDrawStyle
```

Available In: Visual Client Applications

Specifies how stroke (line-drawing) operations will draw on the canvas.

TCanvasElement.TextAlign Property

```
property TextAlign: TTextAlignment
```

Available In: Visual Client Applications

Specifies how text is aligned when it is drawn on the canvas. The default value is taLeftJustify.

TCanvasElement.TextBaseLine Property

```
property TextBaseLine: TTextBaseLine
```

Available In: Visual Client Applications

Specifies the vertical alignment of text when it is drawn on the canvas. The default value is blAlphabetic.

TCanvasElement.Arc Method

```
procedure Arc(X,Y: Double; Radius: Double; StartAngle, EndAngle:
      Double; CounterClockwise: Boolean=False)
```

Available In: Visual Client Applications

Use this method to add an arc to the current subpath of the canvas. The first three parameters specify the center and radius of a circle. The next two parameters are angles (in radians) that specify the start and end points of an arc along the circle. The last parameter specifies whether the arc is traversed counterclockwise or clockwise along the circle's circumference.

TCanvasElement.ArcTo Method

```
procedure ArcTo(X1,Y1,X2,Y2: Double; Radius: Double)
```

Available In: Visual Client Applications

Use this method to add a straight line and an arc to the current subpath of the canvas. The first two parameters specify a starting point and the second two parameters specify an ending point. The arc that is added to the subpath is a portion of a circle with the specified radius. The arc begins and ends at the two specified points, with a line first added to the subpath to connect the current canvas position to the starting point. After drawing the arc, the canvas position is set to the ending point.

TCanvasElement.BeginPath Method

```
procedure BeginPath
```

Available In: Visual Client Applications

Use this method to discard any existing defined subpath for the canvas and begin a new subpath. After this method is called, there is no current point for the canvas.

> **Note**
> This method is implicitly called when the canvas is first created.

TCanvasElement.BezierCurveTo Method

```
procedure BezierCurveTo(CPX1,CPY1,CPX2,CPY2: Double; X,Y:
      Double)
```

Available In: Visual Client Applications

Use this method to add a cubic Bezier curve to the current subpath of the canvas. The first four parameters define the shape of the curve. The starting point of the curve is the current point of the canvas and the end point is defined by the last two parameters. After this method is called, the current point on the canvas is the specified end point.

TCanvasElement.ClearRect Method

```
procedure ClearRect(X,Y: Double; Width,Height: Double)
```

Available In: Visual Client Applications

Use this method to clear the specified rectangle. Clearing the rectangle is equivalent to filling it with a transparent black (clBlack) color.

> **Note**
> This method does not affect the current path or current point of the canvas.

TCanvasElement.Clip Method

```
procedure Clip
```

Available In: Visual Client Applications

Use this method to compute the intersection of the inside of the current path with the current clipping region and use the smaller region as the new clipping region.

Note
Clipping regions cannot be enlarged, so if you only need a temporary clipping region, you should call the Save method to save the current clipping region, create the smaller clipping region, and then restore the previous clipping region by calling the Restore method.

TCanvasElement.ClosePath Method

```
procedure ClosePath
```

Available In: Visual Client Applications

Use this method to close the current subpath of the canvas, if it is open. The subpath is closed by adding a line connecting the current point to the first point of the subpath. A new subpath is then started at the ending point of this last subpath.

Note
The Fill and Clip methods always treat the subpath as closed, so the only time you should need to call this method is when you want to Stroke a closed subpath.

TCanvasElement.ConvertToDataURL Method

```
function ConvertToDataURL(const ImageType: String; ImageQuality:
      Integer=100): String
```

Available In: Visual Client Applications

Use this method to convert the contents of the canvas to an image represented as a Base64-encoded data URL string. Use the ImageType parameter to specify the image format via its MIME type, and the ImageQuality parameter to indicate the image quality for formats that support the specification of the image quality, such as the JPEG format.

Note
Currently, most web browsers only support 'image/png' or 'image/jpeg' as the ImageType parameter. Please consult the web browser documentation on whether additional formats are supported in a specific browser.

TCanvasElement.DrawImage Method

```
procedure DrawImage(Image: TElement; X,Y: Double)

procedure DrawImage(Image: TElement; X,Y: Double; Width,Height:
      Double)

procedure DrawImage(Image: TElement; SourceX,SourceY: Double;
      SourceWidth,SourceHeight: Double; DestX,DestY: Double;
      DestWidth,DestHeight: Double)
```

Available In: Visual Client Applications

Use this method to draw an image, or a portion of an image, at a specified point or, optionally, into a specified rectangle.

The first version of this method simply draws the image at a specified point, retaining the image's original height and width.

The second version of this method draws the image at a specified point, but also scales the image so that it matches the specified height and width.

The last version of this method draws copies the specified rectangle from the image to the specified rectangle on the canvas.

Note
If the specified TElement instance is a TImageElement instance, then it must be completely loaded before passing it to this method. If the image element has not been completely loaded, then an exception will be raised. You can use the Loaded property to determine if an image element has been completely loaded yet.

TCanvasElement.Fill Method

```
procedure Fill
```

Available In: Visual Client Applications

Use this method to fill the current path of the canvas with the color, gradient, or pattern specified by the FillStyle, FillColor, FillGradient, and FillPattern properties.

> **Note**
> If the current subpath is not closed, then this method will treat it as closed. However, it won't actually close the subpath.

TCanvasElement.FillRect Method

```
procedure FillRect(X,Y: Double; Width,Height: Double)
```

Available In: Visual Client Applications

Use this method to fill the specified rectangle with the color, gradient, or pattern specified by the FillStyle, FillColor, FillGradient, and FillPattern properties.

Note
This method does not affect the current path or current point of the canvas.

TCanvasElement.FillText Method

```
procedure FillText(const Text: String; X,Y: Double; MaxWidth:
      Double=-1)
```

Available In: Visual Client Applications

Use this method to draw the specified text at the specified point with the font specified by the Font property, and the color, gradient, or pattern specified by the FillStyle, FillColor, FillGradient, and FillPattern properties.

The optional MaxWidth parameter specifies a maximum width for the text. If the width of the text exceeds this value, then the text is drawn using a condensed font so that it fits within the specified width.

TCanvasElement.IsPointInPath Method

```
function IsPointInPath(X,Y: Double): Boolean
```

Available In: Visual Client Applications

Use this method to determine if the specified point falls within or on the edge of the current path of the canvas.

TCanvasElement.LineTo Method

```
procedure LineTo(X,Y: Double)
```

Available In: Visual Client Applications

Use this method to add a straight line to the current subpath of the canvas, with the current point used as the starting point of the line. After this method is called, the current point of the canvas is set to the specified ending point.

TCanvasElement.MeasureText Method

```
function MeasureText(const Text: String): Double
```

Available In: Visual Client Applications

Use this method to measure the width of the specified text as it would be drawn using the Font property.

TCanvasElement.MoveTo Method

```
procedure MoveTo(X,Y: Double)
```

Available In: Visual Client Applications

Use this method to set the current point of the canvas to the specified point and begin a new subpath at the same point.

> **Note**
> If the previous subpath only consisted of one point, then it is discarded when this method is called.

TCanvasElement.QuadraticCurveTo Method

```
procedure QuadraticCurveTo(CPX,CPY: Double; X,Y: Double)
```

Available In: Visual Client Applications

Use this method to add a quadratic Bezier curve segment to the current subpath of the canvas. The curve starts at the current point of the canvas and ends at the point specified by the last two parameters. The first two parameters specify the shape of the curve between these two points. After this method is called, the current point on the canvas is the specified end point.

TCanvasElement.Rect Method

```
procedure Rect(X,Y: Double; Width, Height: Double)
```

Available In: Visual Client Applications

Use this method to add a rectangle to the current path of the canvas. The added rectangle has its own subpath that is not connected to any other subpaths in the current path. After this method is called, the current point on the canvas is set to the point specified by the first two parameters.

TCanvasElement.Rotate Method

```
procedure Rotate(Angle: Double)
```

Available In: Visual Client Applications

Use this method to change the transformation matrix of the canvas so that any subsequent drawing appears rotated by the specified angle (in radians).

TCanvasElement.Scale Method

```
procedure Scale(SX,SY: Double)
```

Available In: Visual Client Applications

Use this method to change the transformation matrix so that the scale of the canvas is modified according to the specified x and y axis values (independent of one another).

> **Note**
> Specify negative values to "flip" the coordinates of the canvas for a given axis.

TCanvasElement.SetTransform Method

```
procedure SetTransform(M11,M12,M21,M22: Double; DX,DY: Double)
```

Available In: Visual Client Applications

Use this method to directly set the current transformation matrix for the canvas.

TCanvasElement.Stroke Method

```
procedure Stroke
```

Available In: Visual Client Applications

Use this method to stroke the current path of the canvas with the color, gradient, or pattern specified by the StrokeStyle, StrokeColor, StrokeGradient, or StrokePattern properties. The LineCapStyle, LineJoinStyle, MiterLimit and LineWidth properties control how the lines that make up the current path are drawn.

TCanvasElement.StrokeRect Method

```
procedure StrokeRect(X,Y: Double; Width,Height: Double)
```

Available In: Visual Client Applications

Use this method to stroke the specified rectangle with the color, gradient, or pattern specified by the StrokeStyle, StrokeColor, StrokeGradient, or StrokePattern properties. The LineCapStyle, LineJoinStyle, MiterLimit and LineWidth properties control how the lines that make up the rectangle are drawn.

Note
This method does not affect the current path or current point of the canvas.

TCanvasElement.StrokeText Method

```
procedure StrokeText(const Text: String; X,Y: Double; MaxWidth:
      Double=-1)
```

Available In: Visual Client Applications

Use this method draw the outline of the specified text at the specified point with the font specified by the Font property, and the color, gradient, or pattern specified by the StrokeStyle, StrokeColor, StrokeGradient, or StrokePattern properties. The LineCapStyle, LineJoinStyle, MiterLimit and LineWidth properties control how the lines that make up the outline of the font characters are drawn.

The optional MaxWidth parameter specifies a maximum width for the text. If the width of the text exceeds this value, then the text is drawn using a condensed font so that it fits within the specified width.

TCanvasElement.Transform Method

```
procedure Transform(M11,M12,M21,M22: Double; DX,DY: Double)
```

Available In: Visual Client Applications

Use this method to directly modify the existing transformation matrix for the canvas.

TCanvasElement.Translate Method

```
procedure Translate(DX,DY: Double)
```

Available In: Visual Client Applications

Use this method to add horizontal and vertical offsets to the existing transformation matrix for the canvas.

10.33 TCanvasGradient Component

Unit: WebUI

Inherits From TObject

Available In: Visual Client Applications

The TCanvasGradient class represents a linear or radial gradient. It is used by the TCanvasElement class for filling and stroking operations on a canvas.

Properties	Methods	Events
EndRadius	AddColorStop	
EndX	Create	
EndY	RemoveColorStops	
GradientType		
StartRadius		
StartX		
StartY		

TCanvasGradient.EndRadius Property

```
property EndRadius: Double
```

Available In: Visual Client Applications

Specifies the ending radius for the gradient.

> **Note**
> This property only applies to radial gradients (GradientType of gtRadial);

TCanvasGradient.EndX Property

```
property EndX: Double
```

Available In: Visual Client Applications

Specifies the ending x axis point for the gradient.

TCanvasGradient.EndY Property

```
property EndY: Double
```

Available In: Visual Client Applications

Specifies the ending y axis point for the gradient.

TCanvasGradient.GradientType Property

```
property GradientType: TGradientType
```

Available In: Visual Client Applications

Specifies the type of gradient, linear or radial, that is required for the fill or stroke operations on the canvas. The default gradient type is gtLinear.

TCanvasGradient.StartRadius Property

```
property StartRadius: Double
```

Available In: Visual Client Applications

Specifies the starting radius for the gradient.

> **Note**
> This property only applies to radial gradients (GradientType of gtRadial);

TCanvasGradient.StartX Property

```
property StartX: Double
```

Available In: Visual Client Applications

Specifies the starting x axis point for the gradient.

TCanvasGradient.StartY Property

```
property StartY: Double
```

Available In: Visual Client Applications

Specifies the starting y axis point for the gradient.

TCanvasGradient.AddColorStop Method

```
procedure AddColorStop(Position: Double; Color: TColor)
```

Available In: Visual Client Applications

Use this method to add a color stop to the gradient. Color stops are specified as a position between 0.0 and 1.0 along with a color that will start at that position. After specifying two or more color stops, the gradient will smoothly interpolate colors between the various stops.

> **Note**
> If you only specify a single color stop, then the gradient will display that color as a solid color with no interpolation.

TCanvasGradient.Create Method

```
constructor Create(ACanvas: TCanvasElement)
```

Available In: Visual Client Applications

Use this method to create a new instance of the TCanvasGradient class. The ACanvasElement parameter indicates the canvas element that will contain the instance.

TCanvasGradient.RemoveColorStops Method

```
procedure RemoveColorStops
```

Available In: Visual Client Applications

Use this method to remove all color stops for the gradient.

10.34 TCanvasPattern Component

Unit: WebUI

Inherits From TObject

Available In: Visual Client Applications

The TCanvasPattern class represents a pattern comprised of a single image or an image that is tiled horizontally, vertically, or both. It is used by the TCanvasElement class for filling and stroking operations on a canvas.

Properties	Methods	Events
Image	Create	
RepeatStyle		

TCanvasPattern.Image Property

```
property Image: TElement
```

Available In: Visual Client Applications

Specifies the image to use as the pattern for the fill or stroke operations on a canvas.

TCanvasPattern.RepeatStyle Property

```
property RepeatStyle: TPatternRepeatStyle
```

Available In: Visual Client Applications

Specifies how the image specified by the Image property should be tiled when the pattern is used for filling and stroking operations.

TCanvasPattern.Create Method

```
constructor Create(ACanvas: TCanvasElement)
```

Available In: Visual Client Applications

Use this method to create a new instance of the TCanvasPattern class. The ACanvasElement parameter indicates the canvas element that will contain the instance.

10.35 TCanvasPoint Component

Unit: WebUI

Inherits From TObject

Available In: Visual Client Applications

The TCanvasPoint class represents the X/Y floating-point coordinates of a canvas point. It is useful when used with the TCanvasElement class for calculating coordinates for drawing and fill operations on a canvas.

Properties	Methods	Events
X	Assign	
Y	Clear	
	Create	
	Equals	
	Offset	

TCanvasPoint.X Property

```
property X: Double
```

Available In: Visual Client Applications

Specifies the horizontal position of the point.

TCanvasPoint.Y Property

```
property Y: Double
```

Available In: Visual Client Applications

Specifies the vertical position of the point.

TCanvasPoint.Assign Method

```
procedure Assign(APoint: TCanvasPoint)

procedure Assign(AX,AY: Double)
```

Available In: Visual Client Applications

Use this method to assign a source point to the point instance.

TCanvasPoint.Clear Method

```
procedure Clear
```

Available In: Visual Client Applications

Use this method to set both of the point coordinates to 0.

TCanvasPoint.Create Method

```
constructor Create(AX,AY: Double)
```

Available In: Visual Client Applications

Use this method to create a new instance of the TCanvasPoint class using the provided X and Y coordinates.

TCanvasPoint.Equals Method

```
function Equals(APoint: TCanvasPoint): Boolean
```

Available In: Visual Client Applications

Use this method to test if two points have the same coordinates.

TCanvasPoint.Offset Method

```
procedure Offset(AX,AY: Double)
```

Available In: Visual Client Applications

Use this method to offset the point. Offsetting a point increments or decrements its X and Y properties using the AX and AY parameters, respectively.

10.36 TCanvasRect Component

Unit: WebUI

Inherits From TObject

Available In: Visual Client Applications

The TCanvasRect class represents a canvas rectangle using floating-point coordinates. It is useful when used with the TCanvasElement class for calculating rectangle coordinates for drawing and fill operations on a canvas.

Properties	Methods	Events
Bottom	Anchor	
Empty	Assign	
Height	Clear	
Left	Create	
Right	Equals	
Top	Fit	
Width	Inflate	
	Interpolate	
	Normalize	
	Offset	
	Scale	
	Union	

TCanvasRect.Bottom Property

```
property Bottom: Double
```

Available In: Visual Client Applications

Specifies the bottom Y coordinate of the rectangle.

TCanvasRect.Empty Property

```
property Empty: Boolean
```

Available In: Visual Client Applications

Indicates whether the rectangle is empty. A rectangle is considered empty if its width or height is 0.

TCanvasRect.Height Property

```
property Height: Double
```

Available In: Visual Client Applications

Indicates the height of the rectangle (read-only).

TCanvasRect.Left Property

```
property Left: Double
```

Available In: Visual Client Applications

Specifies the left X coordinate of the rectangle.

TCanvasRect.Right Property

```
property Right: Double
```

Available In: Visual Client Applications

Specifies the right X coordinate of the rectangle.

TCanvasRect.Top Property

```
property Top: Double
```

Available In: Visual Client Applications

Specifies the top Y coordinate of the rectangle.

TCanvasRect.Width Property

```
property Width: Double
```

Available In: Visual Client Applications

Indicates the width of the rectangle (read-only).

TCanvasRect.Anchor Method

```
procedure Anchor
```

Available In: Visual Client Applications

Use this method to anchor the rectangle. Anchoring a rectangle changes its Left and Top properties to 0, and adjusts the Right and Bottom properties accordingly so that the rectangle retains its same Width and Height.

TCanvasRect.Assign Method

```
procedure Assign(ARect: TCanvasRect)

procedure Assign(ALeft,ATop,ARight,ABottom: Double)
```

Available In: Visual Client Applications

Use this method to assign a source rectangle, or source rectangle coordinates, to the rectangle instance.

TCanvasRect.Clear Method

```
procedure Clear
```

Available In: Visual Client Applications

Use this method to set all of the rectangle coordinates to 0.

TCanvasRect.Create Method

```
constructor Create(ALeft,ATop,ARight,ABottom: Double=0)
```

Available In: Visual Client Applications

Use this method to create a new instance of the TCanvasRect class. The optional ALeft, ATop, ARight, and ABottom parameters will initialize the instance with the provided coordinates.

TCanvasRect.Equals Method

```
function Equals(ARect: TCanvasRect): Boolean
```

Available In: Visual Client Applications

Use this method to test if two rectangles have the same coordinates.

> **Note**
> If the compared coordinates are not whole numbers and contain fractions, then the comparison results may not be entirely as expected due to the possibility of unequal fractional portions for each coordinate.

TCanvasRect.Fit Method

```
procedure Fit(AWidth,AHeight: Double)
```

Available In: Visual Client Applications

Use this method to proportionally adjust the width and height of the rectangle so that it fits within the specified AWidth and AHeight parameters.

TCanvasRect.Inflate Method

```
procedure Inflate(ALeft,ATop,ARight,ABottom: Double)
```

Available In: Visual Client Applications

Use this method to inflate the rectangle. Inflating a rectangle increments or decrements its Left, Top, Right, and Bottom properties using the specified parameters.

TCanvasRect.Interpolate Method

```
procedure Interpolate(ARect: TCanvasRect; AAmount: Double)
```

Available In: Visual Client Applications

Use this method to calculate a new rectangle using the difference between the coordinates of the rectangle and a source rectangle multiplied by a specified distance amount. This calculation is useful for moving a rectangle along a given linear path between a source rectangle and a destination rectangle.

TCanvasRect.Normalize Method

```
procedure Normalize
```

Available In: Visual Client Applications

Use this method to normalize the rectangle. Normalizing a rectangle changes (if necessary) its Left, Top, Right, and Bottom properties so that the Left property is less than or equal to the Right property and the Top property is less than or equal to the Bottom property.

TCanvasRect.Offset Method

```
procedure Offset(ALeft: Double; ATop: Double)
```

Available In: Visual Client Applications

Use this method to offset the rectangle. Offsetting a rectangle increments or decrements its Left and Top properties using the ALeft and ATop parameters, respectively.

TCanvasRect.Scale Method

```
procedure Scale(ARatio: Double)
```

Available In: Visual Client Applications

Use this method to proportionally scale the coordinates of the rectangle using the specified ARatio parameter.

TCanvasRect.Union Method

```
procedure Union(ARect: TCanvasRect)
```

Available In: Visual Client Applications

Use this method to union the coordinates of the rectangle with another rectangle. A union operation is a max operation on each pair of coordinates, taking the smallest of the two rectangles' Left and Top properties, and the largest of the two rectangles' Right and Bottom properties.

10.37 TCaptionBarControl Component

Unit: WebCtnrs

Inherits From TControl

Available In: Visual Client Applications

The TCaptionBarControl control is the base class for caption bar controls, and contains all of the core panel functionality in the form of public methods and protected properties/events that descendant classes can use to create customized caption bar controls.

Properties	Methods	Events

10.38 TCheckBox Component

Unit: WebBtns

Inherits From TStateButtonControl

Available In: Visual Client Applications

The TCheckBox component represents a checkbox control. A checkbox control is a state control that allows the user to toggle a selection state on and off by using a mouse click, or by pushing the spacebar or enter key.

Properties	Methods	Events
AutoWidth		OnAnimationComplete
Caption		OnAnimationsComplete
Cursor		OnCaptureEnd
DataColumn		OnCaptureStart
DataSet		OnCapturing
Enabled		OnChange
Font		OnClick
Hint		OnContextMenu
ReadOnly		OnEnter
SelectionState		OnExit
TabOrder		OnHide
TabStop		OnKeyDown
ValueSelected		OnKeyPress
ValueUnselected		OnKeyUp
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TCheckBox.AutoWidth Property

```
property AutoWidth: Boolean
```

Available In: Visual Client Applications

Specifies whether the width of the check box should be automatically set based upon the Caption and Font properties.

TCheckBox.Caption Property

```
property Caption: String
```

Available In: Visual Client Applications

Specifies the caption for the control.

> **Note**
> The caption area of a control is also clickable with the mouse or touch interface.

TCheckBox.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TCheckBox.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TCheckBox.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

TCheckBox.Enabled Property

```
property Enabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TCheckBox.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TCheckBox.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TCheckBox.ReadOnly Property

```
property ReadOnly: Boolean
```

Available In: Visual Client Applications

Specifies whether the control's input value can be modified by the user. The default value is False.

Note
The input value can always be programmatically modified.

TCheckBox.SelectionState Property

```
property SelectionState: TSelectionState
```

Available In: Visual Client Applications

Specifies the selection state of the control.

Note
The ssIndeterminate selection state can only be set programmatically. When the user toggles the selection for the control, it will alternate between the ssSelected and ssUnselected selection states.

TCheckBox.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TCheckBox.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TCheckBox.ValueSelected Property

```
property ValueSelected: String
```

Available In: Visual Client Applications

Specifies the textual value to use for the selected state when reading and writing data to and from the data column that the control is bound to. The default value is "True".

TCheckBox.ValueUnselected Property

```
property ValueUnselected: String
```

Available In: Visual Client Applications

Specifies the textual value to use for the unselected state when reading and writing data to and from the data column that the control is bound to. The default value is "False".

TCheckBox.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TCheckBox.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TCheckBox.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TCheckBox.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TCheckBox.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TCheckBox.OnChange Event

```
property OnChange: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TCheckBox.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TCheckBox.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TCheckBox.OnEnter Event

```
property OnEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control obtains input focus.

TCheckBox.OnExit Event

```
property OnExit: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control loses input focus.

TCheckBox.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TCheckBox.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses a key or key combination.

TCheckBox.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TCheckBox.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user releases a key or key combination.

TCheckBox.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TCheckBox.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TCheckBox.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TCheckBox.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TCheckBox.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TCheckBox.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TCheckBox.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TCheckBox.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TCheckBox.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TCheckBox.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TCheckBox.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TCheckBox.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.39 TCollection Component

Unit: WebCore

Inherits From TPersistent

Available In: Client and Server Applications

The TCollection class represents a list of TCollectionItem descendant class instances, and is used to create lists of instances that can be automatically persisted at design-time and run-time. For example, the standard component library uses the TCollection class as the ancestor for the TDataColumns class so that TDataSet instance columns can be persisted with forms.

> **Note**
> Collections always own all items in the collection and will automatically dispose of all items in the collection when the collection is destroyed.

Properties	Methods	Events
Count	Add	OnChanged
ItemClass	BeginUpdate	
Items	Clear	
	Create	
	Delete	
	EndUpdate	
	FindItemByID	
	GetNames	
	IndexOf	
	Insert	
	Move	

TCollection.Count Property

```
property Count: Integer
```

Available In: Client and Server Applications

Indicates the number of items in the collection.

TCollection.ItemClass Property

```
property ItemClass: TCollectionItemClass
```

Available In: Client and Server Applications

Indicates the TCollectionItemClass class type that the collection will use when creating new items for the collection.

TCollection.Items Property

```
property Items[AIndex: Integer]: TCollectionItem

property Items[const AName: String]: TCollectionItem
```

Available In: Client and Server Applications

Accesses items in the collection by their Index property or by their Name.

TCollection.Add Method

```
function Add: TCollectionItem
```

Available In: Client and Server Applications

Use this method to add a new item to the collection. The return value is the new collection item instance, and its class type is determined by the ItemClass property.

TCollection.BeginUpdate Method

```
procedure BeginUpdate
```

Available In: Client and Server Applications

Use this method to begin a batch update to the collection. Batch updates are useful in situations where many changes need to be made to the collection, and triggering the OnChanged event on every change would result in performance issues. This method is reference-counted and every time it is called, an internal counter is incremented. Every time the EndUpdate method is called, the counter is decremented. Once the counter reaches zero, the OnChanged event will be triggered.

TCollection.Clear Method

```
procedure Clear
```

Available In: Client and Server Applications

Use this method to remove all items from the collection, freeing the item instances in the process.

TCollection.Create Method

```
constructor Create(AItemClass: TCollectionItemClass)
```

Available In: Client and Server Applications

Use this method to create a new instance of the TCollection class. The AItemClass parameter indicates the class type that the collection should use when creating new item instances.

TCollection.Delete Method

```
procedure Delete(AIndex: Integer)
```

Available In: Client and Server Applications

Use this method to delete an item from the collection by its Index property, freeing the item instance in the process.

TCollection.EndUpdate Method

```
procedure EndUpdate
```

Available In: Client and Server Applications

Use this method to end a batch update to the collection. Batch updates are useful in situations where many changes need to be made to the collection, and triggering the OnChanged event on every change would result in performance issues. This method is reference-counted and every time it is called, an internal counter is decremented. Every time the BeginUpdate method is called, the counter is incremented. Once the counter reaches zero, the OnChanged event will be triggered.

TCollection.FindItemByID Method

```
function FindItemByID(AID: Integer): TCollectionItem
```

Available In: Client and Server Applications

Use this method to search for an item in the collection by its ID property. An item's ID is an integer identifier that uniquely identifies each item instance.

TCollection.GetNames Method

```
function GetNames: array of String
```

Available In: Client and Server Applications

Use this method to get a list of the names of the items in the collection.

TCollection.IndexOf Method

```
function IndexOf(AItem: TCollectionItem): Integer

function IndexOf(const AName: String): Integer
```

Available In: Client and Server Applications

Use this method to search for an item in the collection by its Index property.

TCollection.Insert Method

```
function Insert(AIndex: Integer): TCollectionItem
```

Available In: Client and Server Applications

Use this method to insert a new item into the collection at a specific index. The return value is the new collection item instance, and its class type is determined by the ItemClass property.

TCollection.Move Method

```
procedure Move(Source: Integer; Dest: Integer)
```

Available In: Client and Server Applications

Use this method to move an item from one index position to another index position. The item's Index will be updated accordingly.

TCollection.OnChanged Event

```
property OnChanged: TNotifyEvent
```

Available In: Client and Server Applications

This event is triggered when a collection item is added or deleted, or if any collection item's name is changed.

10.40 TCollectionItem Component

Unit: WebCore

Inherits From TPersistent

Available In: Client and Server Applications

The TCollectionItem class represents an item in a TCollection class instance.

Properties	Methods	Events
Collection	Create	
ID		
Index		
Name		
Tag		

TCollectionItem.Collection Property

```
property Collection: TCollection
```

Available In: Client and Server Applications

Indicates the collection instance that the item belongs to.

TCollectionItem.ID Property

```
property ID: Integer
```

Available In: Client and Server Applications

Indicates the unique ID of the item. The ID is automatically assigned by the Collection when the item is created.

TCollectionItem.Index Property

```
property Index: Integer
```

Available In: Client and Server Applications

Indicates the index of the item in the Collection. You can modify this property in order to change the index of the item in the collection.

TCollectionItem.Name Property

```
property Name: TCollectionItemName
```

Available In: Client and Server Applications

Specifies the name of the item. A name is not required to be unique, by default, but TCollection descendant classes may enforce uniqueness of item names.

TCollectionItem.Tag Property

```
property Tag: Integer
```

Available In: Client and Server Applications

Can be used to associate any integer value with the collection item.

TCollectionItem.Create Method

```
constructor Create(ACollection: TCollection)
```

Available In: Client and Server Applications

Use this method to create a new instance of the TCollectionItem class. The ACollection parameter indicates the collection instance that will contain the item instance.

10.41 TComponent Component

Unit: WebCore

Inherits From TPersistent

Available In: Client and Server Applications

The TComponent class represents a component and is the base class of all components (visual and non-visual). This class includes functionality for ownership of other components and event registration/notification, which simplifies component destruction by allowing the TComponent class to automatically destroy all owned components.

Properties	Methods	Events
Component	Create	
ComponentCount	FindComponent	
Data	Register	
Destroying	UnRegister	
Name		
Owner		
Tag		

TComponent.Component Property

```
property Component[Index: Integer]: TComponent
```

Available In: Client and Server Applications

Allows indexed access to all components owned by the component. If a component is created with a specific owner, then the component instance that is created is automatically added to the owner component's Component property. When an owner component is destroyed, all owned components are also automatically destroyed at the same time.

TComponent.ComponentCount Property

```
property ComponentCount: Integer
```

Available In: Client and Server Applications

Indicates the number of components owned by the component.

TComponent.Data Property

```
property Data: TObject
```

Available In: Client and Server Applications

Can be used to associate any TObject or descendant class instance with the component.

TComponent.Destroying Property

```
property Destroying: Boolean
```

Available In: Client and Server Applications

Indicates that the component is in the process of being destroyed, meaning that its Destroy destructor has begun executing.

TComponent.Name Property

```
property Name: TComponentName
```

Available In: Client and Server Applications

Specifies the name of the component. Components that are placed on a form must have a name that is unique within the context of the containing form.

TComponent.Owner Property

```
property Owner: TComponent
```

Available In: Client and Server Applications

Indicates the owner of the component, or nil if the component does not have an owner.

TComponent.Tag Property

```
property Tag: Integer
```

Available In: Client and Server Applications

Can be used to associate any integer value with the component.

TComponent.Create Method

```
constructor Create(AOwner: TComponent)
```

Available In: Client and Server Applications

Use this method to create a new instance of the TComponent class. The AOwner parameter indicates the component that will own the instance, or nil if the component has no owner. The owner of the component will automatically free the component instance when the owner is freed.

TComponent.FindComponent Method

```
function FindComponent(const Value: String; Recurse:
      Boolean=False): TComponent
```

Available In: Client and Server Applications

Use this method to find an owned component by its name. The optional Recurse parameter indicates whether the search should be recursive and include components owned by the owned components, etc.

TComponent.Register Method

```
procedure Register(Component: TComponent)
```

Available In: Client and Server Applications

Registers a component with another component so that the calling component receives component notification messages.

For example, a component can register itself with another component so that it can be notified when the component is destroyed. This is useful in situations where a component instance needs to know when another component instance is destroyed so that it can remove any references to the component instance being destroyed.

TComponent.UnRegister Method

```
procedure UnRegister(Component: TComponent)
```

Available In: Client and Server Applications

Unregisters a component from another component so that the calling component no longer receives any notification messages.

10.42 TConstraint Component

Unit: WebUI

Inherits From TElementAttribute

Available In: Visual Client Applications

The TConstraint class represents a set of width and height constraints for a UI element or control.

Properties	Methods	Events
Height	SetToDefault	
Width		

TConstraint.Height Property

```
property Height: Integer
```

Available In: Visual Client Applications

Specifies the height constraint. A value of zero (the default) indicates that no height constraint is in effect.

TConstraint.Width Property

```
property Width: Integer
```

Available In: Visual Client Applications

Specifies the width constraint. A value of zero (the default) indicates that no width constraint is in effect.

TConstraint.SetToDefault Method

```
procedure SetToDefault
```

Available In: Visual Client Applications

Use this method to reset the constraint's properties to their default values.

10.43 TConstraints Component

Unit: WebUI

Inherits From TElementAttribute

Available In: Visual Client Applications

The TConstraints class represents the minimum/maximum width and height constraints for a UI element or control. Width and height constraints ensure that any attempts to resize the UI element or control is subject to the defined constraints.

Properties	Methods	Events
Max	SetToDefault	
Min		

TConstraints.Max Property

```
property Max: TConstraint
```

Available In: Visual Client Applications

Specifies the maximum width and height constraints.

TConstraints.Min Property

```
property Min: TConstraint
```

Available In: Visual Client Applications

Specifies the minimum width and height constraints.

TConstraints.SetToDefault Method

```
procedure SetToDefault
```

Available In: Visual Client Applications

Use this method to reset the constraints' properties to their default values.

10.44 TContentLayout Component

Unit: WebCtrls

Inherits From TPersistent

Available In: Visual Client Applications

The TContentLayout class represents the layout of content in UI elements, and is used to apply layouts similar to that of background images to a content element in controls like the TImage control.

Properties	Methods	Events
Height		
Left		
Position		
Size		
Top		
Width		

TContentLayout.Height Property

```
property Height: Integer
```

Available In: Visual Client Applications

Specifies the height of the element. If the actual height of the element is different than this value, then the element height is stretched/contracted accordingly.

> **Note**
> This property is only valid when the Size is csSpecified.

TContentLayout.Left Property

```
property Left: Integer
```

Available In: Visual Client Applications

Specifies the left position of the element.

Note
This property is only valid when the Position is cpSpecified.

TContentLayout.Position Property

```
property Position: TContentPosition
```

Available In: Visual Client Applications

Specifies how the element should be positioned within the UI element or control.

TContentLayout.Size Property

```
property Size: TContentSize
```

Available In: Visual Client Applications

Specifies how the element should be sized within the UI element or control.

TContentLayout.Top Property

```
property Top: Integer
```

Available In: Visual Client Applications

Specifies the top position of the element.

> **Note**
> This property is only valid when the Position is cpSpecified.

TContentLayout.Width Property

```
property Width: Integer
```

Available In: Visual Client Applications

Specifies the width of the element. If the actual width of the element is different than this value, then the element width is stretched/contracted accordingly.

> **Note**
> This property is only valid when the Size is csSpecified.

10.45 TControl Component

Unit: WebCtrls

Inherits From TInterfaceController

Available In: Visual Client Applications

The TControl component is the base class for all visual controls and provides layout and dimensional functionality, as well as functionality for handling child controls, mouse, keyboard, and touch events, focus, and visibility.

Properties	Methods	Events
ActiveControl	BeginUpdate	
AlwaysOnTop	BringToFront	
Animations	CanFocus	
ClientHeight	ControlAt	
ClientID	DefineLayout	
ClientWidth	EndUpdate	
Constraints	Float	
ControlCount	ForceUpdate	
Controls	GetControlNames	
DisplayOrder	Hide	
Focused	IndexOfControl	
Height	MakeVisible	
InUpdate	Minimize	
Layout	RemoveFocus	
LayoutOrder	Restore	
Left	ScrollBy	
Margins	SendToBack	
Minimized	SetFocus	
Parent	Show	
ScrollHeight	SlideTo	
ScrollLeft	Tab	
ScrollTop		
ScrollWidth		
SurfaceLeft		
SurfaceTop		
Top		
Visible		
VisibleControlCount		

VisibleControls		
Width		

TControl.ActiveControl Property

```
property ActiveControl: TControl
```

Available In: Visual Client Applications

Indicates which child control, if any, has focus in the current control.

TControl.AlwaysOnTop Property

```
property AlwaysOnTop: Boolean
```

Available In: Visual Client Applications

Specifies that the control should always be visually placed on top of any other control within the same container control.

TControl.Animations Property

```
property Animations: TAnimations
```

Available In: Visual Client Applications

Specifies the animations for the control.

TControl.ClientHeight Property

```
property ClientHeight: Integer
```

Available In: Visual Client Applications

Indicates the height of the client rectangle for the control.

TControl.ClientID Property

```
property ClientID: String
```

Available In: Visual Client Applications

Specifies the unique DOM ID to assign to the control's client element. This is useful for situations where you need to identify the element from external Javascript code. By default, the interface manager never assigns DOM IDs to elements because it doesn't need or use them to identify UI elements.

TControl.ClientWidth Property

```
property ClientWidth: Integer
```

Available In: Visual Client Applications

Indicates the width of the client rectangle for the control.

TControl.Constraints Property

```
property Constraints: TConstraints
```

Available In: Visual Client Applications

Indicates the dimensional constraints for the control.

TControl.ControlCount Property

```
property ControlCount: Integer
```

Available In: Visual Client Applications

Indicates the total number of child controls for the control.

TControl.Controls Property

```
property Controls[AIndex: Integer]: TControl
```

Available In: Visual Client Applications

Accesses a child control by its index position in the child controls, which are ordered by their LayoutOrder property.

TControl.DisplayOrder Property

```
property DisplayOrder: Integer
```

Available In: Visual Client Applications

Specifies the display order, or visual stacking order, of the control within its parent container control.

TControl.Focused Property

```
property Focused: Boolean
```

Available In: Visual Client Applications

Indicates whether the control currently has focus.

TControl.Height Property

```
property Height: Integer
```

Available In: Visual Client Applications

Indicates the current height of the control, and can be used to specify the **defined** height for the control. The defined height is the last value that was directly assigned to the Height property. Layout property changes may affect the value returned by the Height property.

TControl.InUpdate Property

```
property InUpdate: Boolean
```

Available In: Visual Client Applications

Indicates whether the control, or any of its parent controls, is currently in a batch update. A control is in a batch update if the BeginUpdate method is called on the control or any of its parent controls. When a control is in a batch update, it doesn't apply any changes to any of its properties until the EndUpdate method is called on the control or any of its parent controls, and the InUpdate property returns False.

Note
Updates are reference-counted so calls to the BeginUpdate method increment the reference count, and calls to the EndUpdate method decrement the reference count.

TControl.Layout Property

```
property Layout: TLayout
```

Available In: Visual Client Applications

Specifies the layout for the control.

TControl.LayoutOrder Property

```
property LayoutOrder: Integer
```

Available In: Visual Client Applications

Specifies the layout order of the control. The layout order determines the order in which the child controls of a container control are positioned and sized using the layout management functionality for controls.

TControl.Left Property

```
property Left: Integer
```

Available In: Visual Client Applications

Indicates the current left position of the control, and can be used to specify the **defined** left position for the control. The defined left position is the last value that was directly assigned to the Left property. Layout property changes may affect the value returned by the Left property.

TControl.Margins Property

```
property Margins: TMargins
```

Available In: Visual Client Applications

Specifies the margins to be used for the control when the control is being positioned/sized using the layout management functionality.

TControl.Minimized Property

```
property Minimized: Boolean
```

Available In: Visual Client Applications

Indicates whether the control is minimized. A control can be minimized using the Minimize method, and restored using the Restore method.

TControl.Parent Property

```
property Parent: TControl
```

Available In: Visual Client Applications

Specifies the parent control that contains the control, or nil if the control has no parent.

TControl.ScrollHeight Property

```
property ScrollHeight: Integer
```

Available In: Visual Client Applications

Indicates the total height of the control's content and/or its child controls. If a control's content height is greater than its Height property, then you can use the ScrollTop property to programmatically scroll the control vertically, or to find out the current vertical scroll position.

TControl.ScrollLeft Property

```
property ScrollLeft: Integer
```

Available In: Visual Client Applications

Specifies the horizontal scroll position for the control. If a control's ScrollWidth property is greater than its Width property, then you can use this property to programmatically scroll the control horizontally, or to find out the current horizontal scroll position.

TControl.ScrollTop Property

```
property ScrollTop: Integer
```

Available In: Visual Client Applications

Specifies the vertical scroll position for the control. If a control's ScrollHeight property is greater than its Height property, then you can use this property to programmatically scroll the control vertically, or to find out the current vertical scroll position.

TControl.ScrollWidth Property

```
property ScrollWidth: Integer
```

Available In: Visual Client Applications

Indicates the total width of the control's content and/or its child controls. If a control's content width is greater than its Width property, then you can use the ScrollLeft property to programmatically scroll the control horizontally, or to find out the current horizontal scroll position.

TControl.SurfaceLeft Property

```
property SurfaceLeft: Integer
```

Available In: Visual Client Applications

Indicates the current left position of the control, relative to the application surface.

The application surface is represented by the Surface property of the global Application variable in the WebForms unit.

TControl.SurfaceTop Property

```
property SurfaceTop: Integer
```

Available In: Visual Client Applications

Indicates the current top position of the control, relative to the application surface.

The application surface is represented by the Surface property of the global Application variable in the WebForms unit.

TControl.Top Property

```
property Top: Integer
```

Available In: Visual Client Applications

Indicates the current top position of the control, and can be used to specify the **defined** top position for the control. The defined top position is the last value that was directly assigned to the Top property. Layout property changes may affect the value returned by the Top property.

TControl.Visible Property

```
property Visible: Boolean
```

Available In: Visual Client Applications

Indicates whether the control is visible or not. Setting this property to False causes the Hide method to be called, whereas setting this property to True causes the Show method to be called. The default value is True.

TControl.VisibleControlCount Property

```
property VisibleControlCount: Integer
```

Available In: Visual Client Applications

Indicates the total number of visible child controls for the control. A visible child control is one whose Visible property is True.

TControl.VisibleControls Property

```
property VisibleControls[AIndex: Integer]: TControl
```

Available In: Visual Client Applications

Accesses a visible child control by its index position in the child controls, which are ordered by their LayoutOrder property.

TControl.Width Property

```
property Width: Integer
```

Available In: Visual Client Applications

Indicates the current width of the control, and can be used to specify the **defined** width for the control. The defined width is the last value that was directly assigned to the Width property. Layout property changes may affect the value returned by the Width property.

TControl.BeginUpdate Method

```
procedure BeginUpdate
```

Available In: Visual Client Applications

Use this method to begin a batch update on a control. A control is in a batch update if the BeginUpdate method is called on the control, or any of its parent controls. When a control is in a batch update, it doesn't apply any changes to any of its properties until the EndUpdate method is called on the control or any of its parent controls, and the InUpdate property returns False.

> **Note**
> Updates are reference-counted so calls to the BeginUpdate method increment the reference count, and calls to the EndUpdate method decrement the reference count.

TControl.BringToFront Method

```
procedure BringToFront
```

Available In: Visual Client Applications

Use this method to bring the control to the front of the visual stacking order. A control in the front will have a DisplayOrder property equal to one less than the number of child controls in the parent container control.

TControl.CanFocus Method

```
function CanFocus: Boolean
```

Available In: Visual Client Applications

Use this method to determine if the current control can acquire input focus. Invisible and disabled controls always return False when this method is called.

TControl.ControlAt Method

```
function ControlAt(X,Y: Integer): TControl
```

Available In: Visual Client Applications

Use this method to find the control, if one exists, at the specified coordinates. The search is performed using the visual stacking order defined by the DisplayOrder property of each control. Controls with a smaller display order (in front of other controls) will be found instead of controls with a larger display order, even if both controls have the same bounds.

TControl.DefineLayout Method

```
procedure DefineLayout
```

Available In: Visual Client Applications

Use this method to assign the current bounds of the control to the Left, Top, Width, and Height properties of the control.

This method is useful in situations where a control has been positioned or sized according to its Layout properties, but you wish to have the bounds persist even after modifying the layout properties so that they no longer position or size the control in the same way.

TControl.EndUpdate Method

```
procedure EndUpdate
```

Available In: Visual Client Applications

Use this method to end a batch update on a control. A control is in a batch update if the BeginUpdate method is called on the control or any of its parent controls. When a control is in a batch update, it doesn't apply any changes to any of its properties until the EndUpdate method is called on the control or any of its parent controls, and the InUpdate property returns False.

Note
Updates are reference-counted so calls to the BeginUpdate method increment the reference count, and calls to the EndUpdate method decrement the reference count.

TControl.Float Method

```
procedure Float
```

Available In: Visual Client Applications

Use this method to parent the control to the application surface and reset any layout properties set for the control. This method is useful as the first step when dragging controls around the application surface.

The application surface is represented by the Surface property of the global Application variable in the WebForms unit.

TControl.ForceUpdate Method

```
procedure ForceUpdate
```

Available In: Visual Client Applications

Use this method to force the control to perform a layout update when in the middle of a BeginUpdate/EndUpdate block. Normally, a control will not update its layout bounds when an update block is in effect. A forced layout update will ensure that the control's layout bounds are updated to reflect any layout changes applied to the control within the update block.

TControl.GetControlNames Method

```
function GetControlNames(AControlClass: TControlClass): array of
      String
```

Available In: Visual Client Applications

Use method to get a list of the names of the child controls for the control.

TControl.Hide Method

```
procedure Hide
```

Available In: Visual Client Applications

Use this method to hide the control. After calling this method, the Visible property will be set to False.

TControl.IndexOfControl Method

```
function IndexOfControl(AControl: TControl; AControlClass:
      TControlClass=nil): Integer
```

Available In: Visual Client Applications

Use this method to determine the index of a child control within the Controls property. You can, optionally, also pass a TControlClass type to limit the search to a particular TControl class type.

TControl.MakeVisible Method

```
procedure MakeVisible(X,Y: Boolean=True)
```

Available In: Visual Client Applications

Use this method to ensure that the control is visible within the client rectangle of its parent control.

TControl.Minimize Method

```
procedure Minimize
```

Available In: Visual Client Applications

Use this method to minimize the control. The minimization of a control is a control-specific operation, so the results of calling this method will depend upon the control class of the control being minimized.

TControl.RemoveFocus Method

```
procedure RemoveFocus
```

Available In: Visual Client Applications

Use this method to remove focus from the control. If the control is not focused, then this method does nothing.

TControl.Restore Method

```
procedure Restore
```

Available In: Visual Client Applications

Use this method to restore a minimized control back to its original dimensions.

TControl.ScrollBy Method

```
procedure ScrollBy(X,Y: Integer)
```

Available In: Visual Client Applications

If a control's content width and/or height is greater than its Width and Height properties, then you can use this method to scroll the contents of the control horizontally, vertically, or both. The X and Y values represent the number of pixels by which to scroll the contents, and may be negative values for scrolling backward.

TControl.SendToBack Method

```
procedure SendToBack
```

Available In: Visual Client Applications

Use this method to send the control to the back of the visual stacking order. A control in the back will have a DisplayOrder property equal to 0.

TControl.SetFocus Method

```
procedure SetFocus
```

Available In: Visual Client Applications

Use this method to set focus to the control. Use the Focused property to determine if the control was actually able to obtain the focus.

The ability of a control to obtain the input focus is dependent upon:

- Whether the control is focusable
- Whether the control is visible
- Whether its parent container control is visible

TControl.Show Method

```
procedure Show
```

Available In: Visual Client Applications

Use this method to show the control. After calling this method, the Visible property will be set to True.

TControl.SlideTo Method

```
procedure SlideTo(X,Y: Integer; AnimationStyle: TAnimationStyle;
      AnimationDuration: Integer; FadeInOut: Boolean=False)
```

Available In: Visual Client Applications

Use this method to slide the control to a specific location using a specific animation style/duration and, optionally, with a corresponding fade (hides the control).

> **Note**
> This method updates the properties of the Left, Top, and, optionally, the Visible animation primitives for the control's Animations property.

TControl.Tab Method

```
procedure Tab(Backward: Boolean=False)
```

Available In: Visual Client Applications

Use this method to navigate to the next or prior control after/before the currently-focused control, based upon the tabbing order for the container control.

The Backward parameter determines if the navigation occurs forwards or backwards in the tabbing order.

10.46 TCookies Component

Unit: WebComps

Inherits From TObject

Available In: Client Applications

The TCookies class encapsulates the cookie functionality in the web browser.

> **Note**
> The component library includes a global instance variable of this class called Cookies in the WebComps unit that should be used instead of creating new instances of the class.

> **Warning**
> Do not attempt to store more than 4k of data in a single cookie (name and value combined). If you need to store more data than that, then you should use the Local Storage functionality instead.

Properties	Methods	Events
Count	Clear	
Enabled	Create	
Items	Exists	
	Refresh	
	Set	

TCookies.Count Property

```
property Count: Integer
```

Available In: Client Applications

Indicates the number of cookies defined.

TCookies.Enabled Property

```
property Enabled: Boolean
```

Available In: Client Applications

Indicates whether cookies are enabled or not in the web browser. If this property is False, then the cookie functionality is disabled and the TCookies object will not function properly.

TCookies.Items Property

```
property Items[Index: Integer]: String

property Items[const AName: String]: String
```

Available In: Client Applications

Accesses all defined cookies by index or by name.

TCookies.Clear Method

```
procedure Clear(const AName: String; const Path: String='';
      const Domain: String=''; Secure: Boolean=False)
```

Available In: Client Applications

Use this method to clear an existing cookie. The Name parameter is the cookie name, the Path parameter is the relative path under the domain that the cookie applies to, the Domain parameter is the web site domain that the cookie applies to, and the Secure parameter indicates whether the cookie applies to a secure domain/path (https://) or a normal domain/path (http://).

Note
Use the Exists method to determine if a cookie exists before trying to clear the cookie.

TCookies.Create Method

```
constructor Create
```

Available In: Client Applications

Use this method to create a new instance of the TCookies class.

TCookies.Exists Method

```
function Exists(const AName: String): Boolean
```

Available In: Client Applications

Use this method to determine if the specified cookie exists.

TCookies.Refresh Method

```
procedure Refresh
```

Available In: Client Applications

Use this method to reload the cookies from the web browser. This is useful for situations where an embedded HTML document in a TBrowser control has modified the cookies and these modifications need to be reflected in the global TCookies instance.

Note
Cookies are cached in a TCookies instance and do not immediately reflect changes to the web browser cookies done via third party JavaScript code or embedded documents. This is why this method is necessary.

TCookies.Set Method

```
procedure Set(const AName: String; const Value: String; MaxAge:
      Integer=-1; const Path: String=''; const Domain: String='';
      Secure: Boolean=False)
```

Available In: Client Applications

Use this method to set a cookie value. The Name parameter is the cookie name, the MaxAge parameter specifies the length of time, in seconds, that the browser should cache the cookie, the Path parameter is the relative path under the domain that the cookie applies to, the Domain parameter is the web site domain that the cookie applies to, and the Secure parameter indicates whether the cookie applies to a secure domain/path (https://) or a normal domain/path (http://).

Note
Using -1 (or any value less than 0) as the MaxAge parameter causes the cookie to be a session-only cookie that is automatically deleted by the web browser when it is closed/shut down.

10.47 TCorner Component

Unit: WebUI

Inherits From TElementAttribute

Available In: Visual Client Applications

The TCorners class represents the horizontal and vertical radius of a border corner for a UI element or control.

Properties	Methods	Events
HorzRadius	SetToDefault	
VertRadius		

TCorner.HorzRadius Property

```
property HorzRadius: Integer
```

Available In: Visual Client Applications

Specifies the horizontal radius.

TCorner.VertRadius Property

```
property VertRadius: Integer
```

Available In: Visual Client Applications

Specifies the vertical radius.

TCorner.SetToDefault Method

```
procedure SetToDefault
```

Available In: Visual Client Applications

Use this method to reset the corner's properties to their default values.

10.48 TCorners Component

Unit: WebUI

Inherits From TElementAttribute

Available In: Visual Client Applications

The TCorner class represents the horizontal and vertical radii of the border corners for a UI element or control.

Properties	Methods	Events
BottomLeft	SetToDefault	
BottomRight		
TopLeft		
TopRight		

TCorners.BottomLeft Property

```
property BottomLeft: TCorner
```

Available In: Visual Client Applications

Specifies the bottom-left corner radius.

TCorners.BottomRight Property

```
property BottomRight: TCorner
```

Available In: Visual Client Applications

Specifies the bottom-right corner radius.

TCorners.TopLeft Property

```
property TopLeft: TCorner
```

Available In: Visual Client Applications

Specifies the top-left corner radius.

TCorners.TopRight Property

```
property TopRight: TCorner
```

Available In: Visual Client Applications

Specifies the top-right corner radius.

TCorners.SetToDefault Method

```
procedure SetToDefault
```

Available In: Visual Client Applications

Use this method to reset the corners' properties to their default values.

10.49 TDatabase Component

Unit: WebData

Inherits From TComponent

Available In: Client and Server Applications

The TDatabase component represents a database container for all datasets in an application and provides properties and methods for loading/saving datasets and using transactions to modify the datasets. An instance of the TDatabase component called **Database** is automatically created by the component library at application startup as the default, global TDatabase instance.

Properties	Methods	Events
ActiveServerSession	CancelPendingRequests	AfterCommit
AutoTransactions	Commit	AfterRollback
BaseURL	LoadColumns	BeforeCommit
DatabaseName	LoadRows	BeforeRollback
DataSetCount	RetryPendingRequests	OnCommitError
DataSets	Rollback	OnCreate
InTransaction	StartTransaction	OnDestroy
NumPendingRequests		OnRollbackError
Params		
ServerSession		
Timeout		
TransactionLevel		

TDatabase.ActiveServerSession Property

```
property ActiveServerSession: TServerSession
```

Available In: Client Applications

Indicates the TServerSession instance being used by the database for authentication. If no session has been manually assigned to the ServerSession property, then this property will reference the global **Session** instance that is automatically created at application startup.

The database uses the active server session's UserName, Password, and BaseURL properties along with the Authenticate method to authenticate against the web server before executing any database API requests. The database then uses the BaseURL and DatabasesResource properties for constructing database API requests.

TDatabase.AutoTransactions Property

```
property AutoTransactions: Boolean
```

Available In: Client and Server Applications

Specifies whether transactions will be automatically started and committed or rolled back as rows are inserted, updated, and deleted in the datasets owned by the database. The default value is True.

Note
Automatic transactions are implicitly nested. If you insert a row in one dataset, and then edit a row in another dataset, the TransactionLevel property will be 1.

TDatabase.BaseURL Property

```
property BaseURL: String
```

Available In: Client Applications

Indicates the base URL used for all database API requests. This URL is a combination of:

- The ActiveServerSession BaseURL property
- The ActiveServerSession DatabasesResource property
- The DatabaseName property

This property is used by datasets to build the proper URLs for loading rows and BLOB data from the web server.

TDatabase.DatabaseName Property

```
property DatabaseName: String
```

Available In: Client and Server Applications

Specifies the name of the database to use when interacting with the database API.

TDatabase.DataSetCount Property

```
property DataSetCount: Integer
```

Available In: Client and Server Applications

Indicates the number of datasets owned by this database.

TDatabase.DataSets Property

```
property DataSets[Index: Integer]: TDataSet

property DataSets[const Name: String]: TDataSet
```

Available In: Client and Server Applications

Accesses all datasets owned by this database by index or by name.

TDatabase.InTransaction Property

```
property InTransaction: Boolean
```

Available In: Client and Server Applications

Indicates whether a transaction is active or not.

TDatabase.NumPendingRequests Property

```
property NumPendingRequests: Integer
```

Available In: Client Applications

Indicates the number of pending database load/commit web server requests. If any errors were encountered during a dataset columns/parameters/rows load or database transaction commit operation, then the web server request used with the operation will remain as a pending request. You can use the RetryPendingRequests to retry all pending requests, or the CancelPendingRequests to cancel all pending requests.

TDatabase.Params Property

```
property Params: TStrings
```

Available In: Client and Server Applications

Specifies any custom database-specific parameters to use with any dataset commands for the datasets in the database. These parameters are included with any relevant database API requests and are useful for specifying additional parameters for the web server to use when executing dataset commands.

TDatabase.ServerSession Property

```
property ServerSession: TServerSession
```

Available In: Client Applications

Specifies the TServerSession instance to use with the database. The default value is nil.

The ActiveServerSession property refers to the server session that is actually being used by the database for authentication and database API requests. If the ServerSession property is nil, then the ActiveServerSession property will reference the global **Session** instance that is automatically created at application startup.

TDatabase.Timeout Property

```
property Timeout: Integer
```

Available In: Client and Server Applications

Specifies how long any database request should wait, in seconds, for a successful connection to the server before returning an error. The default value is 0, which means to wait a browser-defined number of seconds.

TDatabase.TransactionLevel Property

```
property TransactionLevel: Integer
```

Available In: Client and Server Applications

Indicates the current transaction level. A value of -1 means that no transaction is active, while a value of 0 or higher indicates that a transaction is active up to N levels of nesting.

TDatabase.CancelPendingRequests Method

```
procedure CancelPendingRequests
```

Available In: Client Applications

Use this method to cancel any pending dataset columns/parameters/rows load or transaction commit requests. You can determine if there are any pending requests by examining the NumPendingRequests property.

TDatabase.Commit Method

```
procedure Commit
```

Available In: Client and Server Applications

Use this method to commit the active transaction.

Warning
For client applications, transaction commits are asynchronous and the Commit method may run to completion before the AfterCommit or OnCommitError event handlers are executed.

TDatabase.LoadColumns Method

```
procedure LoadColumns(DataSet: TDataSet)
```

Available In: Client and Server Applications

Use this method to load the columns for the specified dataset from the web server. This method will replace all existing columns in the specified dataset with the current columns for the web server dataset.

TDatabase.LoadRows Method

```
procedure LoadRows(DataSet: TDataSet; Append: Boolean=False)
```

Available In: Client and Server Applications

Use this method to load the rows for the specified dataset from the web server. Specify True for the Append parameter to have the rows appended to the dataset. The default behavior is to replace all rows in the dataset with the rows returned by the web server.

TDatabase.RetryPendingRequests Method

```
procedure RetryPendingRequests
```

Available In: Client Applications

Use this method to retry any pending dataset columns/parameters/rows load or transaction commit requests. You can determine if there are any pending requests by examining the NumPendingRequests property.

TDatabase.Rollback Method

```
procedure Rollback
```

Available In: Client and Server Applications

Use this method to roll back the active transaction.

TDatabase.StartTransaction Method

```
procedure StartTransaction
```

Available In: Client and Server Applications

Use this method to start a new transaction.

TDatabase.AfterCommit Event

```
property AfterCommit: TNotifyEvent
```

Available In: Client and Server Applications

This event is triggered after the Commit method completes.

Warning
For client applications, transaction commits are asynchronous and the AfterCommit event handler may not be fired until after the Commit method runs to completion.

TDatabase.AfterRollback Event

```
property AfterRollback: TNotifyEvent
```

Available In: Client and Server Applications

This event is triggered after the Rollback method completes.

TDatabase.BeforeCommit Event

```
property BeforeCommit: TDatabaseEvent
```

Available In: Client and Server Applications

This event is triggered before the Commit method starts. Return False from the event handler to prevent the commit from occurring.

TDatabase.BeforeRollback Event

```
property BeforeRollback: TDatabaseEvent
```

Available In: Client and Server Applications

This event is triggered before the Rollback method starts. Return False from the event handler to prevent the rollback from occurring.

TDatabase.OnCommitError Event

```
property OnCommitError: TDatabaseErrorEvent
```

Available In: Client and Server Applications

This event is triggered whenever an error occurs during the execution of the Commit method.

Note
If an event handler is not assigned to this event, then the error will be raised as an exception.

Warning
For client applications, transaction commits are asynchronous and the OnCommitError event handler may not be fired until after the Commit method runs to completion.

TDatabase.OnCreate Event

```
property OnCreate: TNotifyEvent
```

Available In: Client and Server Applications

This event is triggered after the database is created and initialized.

Note
Any design-time components placed on the database will already be instantiated and initialized before this event is triggered.

TDatabase.OnDestroy Event

```
property OnDestroy: TNotifyEvent
```

Available In: Client and Server Applications

This event is triggered before the database is destroyed. Use this event to dispose of any instances or resources that may have been allocated in the OnCreate event handler.

TDatabase.OnRollbackError Event

```
property OnRollbackError: TDatabaseErrorEvent
```

Available In: Client and Server Applications

This event is triggered whenever an error occurs during the execution of the Rollback method.

Note
If an event handler is not assigned to this event, then the error will be raised as an exception.

10.50 TDataColumn Component

Unit: WebData

Inherits From TCollectionItem

Available In: Client and Server Applications

The TDataColumn class represents a column in a TDataSet component and contains functionality for getting and setting column values, tracking modifications, and sorting.

Properties	Methods	Events
AsBlob	Clear	OnGetText
AsBoolean		OnSetText
AsDate		
AsDateTime		
AsFloat		
AsInteger		
AsString		
AsTime		
Calculated		
DataType		
Length		
Modified		
Null		
OldValue		
ReadOnly		
Scale		
SortDirection		
Text		

TDataColumn.AsBlob Property

```
property AsBlob: TStream
```

Available In: Server Applications

Gets or sets the column value in the active row using a TStream class interface. If the column value cannot be expressed as a BLOB stream, then an exception will be raised.

TDataColumn.AsBoolean Property

```
property AsBoolean: Boolean
```

Available In: Client and Server Applications

Gets or sets the column value in the active row as a boolean value. If the column value cannot be expressed as a boolean value, then an exception will be raised.

TDataColumn.AsDate Property

```
property AsDate: DateTime
```

Available In: Client and Server Applications

Gets or sets the column value in the active row as a date value. If the column value cannot be expressed as a date value, then an exception will be raised.

TDataColumn.AsDateTime Property

```
property AsDateTime: DateTime
```

Available In: Client and Server Applications

Gets or sets the column value in the active row as a date/time value. If the column value cannot be expressed as a date/time value, then an exception will be raised.

TDataColumn.AsFloat Property

```
property AsFloat: Double
```

Available In: Client and Server Applications

Gets or sets the column value in the active row as a floating-point value. If the column value cannot be expressed as a floating-point value, then an exception will be raised.

TDataColumn.AsInteger Property

```
property AsInteger: Integer
```

Available In: Client and Server Applications

Gets or sets the column value in the active row as an integer value. If the column value cannot be expressed as an integer value, then an exception will be raised.

TDataColumn.AsString Property

```
property AsString: String
```

Available In: Client and Server Applications

Gets or sets the column value in the active row as a string value. If the column value cannot be expressed as a string value, then an exception will be raised.

TDataColumn.AsTime Property

```
property AsTime: DateTime
```

Available In: Client and Server Applications

Gets or sets the column value in the active row as a time value. If the column value cannot be expressed as a time value, then an exception will be raised.

TDataColumn.Calculated Property

```
property Calculated: Boolean
```

Available In: Client and Server Applications

Specifies that the column is calculated via the TDataSet OnCalculateRow event handler, if one exists.

Note
Any attempt to programmatically modify a calculated column outside of an OnCalculateRow event handler will result in an error.

TDataColumn.DataType Property

```
property DataType: TDataType
```

Available In: Client and Server Applications

Gets or sets the data type of the column.

> **Note**
> Attempting to change the data type of a column for a dataset that has been opened will result in an exception.

TDataColumn.Length Property

```
property Length: Integer
```

Available In: Client and Server Applications

Gets or sets the length of the column. The length should only be set for string and BLOB column types and all other column types should have a length of -1.

> **Note**
> Attempting to change the length of a column for a dataset that has been opened will result in an exception.

TDataColumn.Modified Property

```
property Modified: Boolean
```

Available In: Client and Server Applications

Indicates if the column has been modified in the active row as part of an insert or update operation.

TDataColumn.Null Property

```
property Null: Boolean
```

Available In: Client and Server Applications

Indicates if the column in the active row is NULL.

TDataColumn.OldValue Property

```
property OldValue: TDataColumnValue
```

Available In: Client and Server Applications

Provides a reference to the prior version of the column value when updating a row in a dataset.

TDataColumn.ReadOnly Property

```
property ReadOnly: Boolean
```

Available In: Client and Server Applications

Specifies that the column is read-only.

Note
Any attempt to programmatically modify a read-only column will result in an error.

TDataColumn.Scale Property

```
property Scale: Integer
```

Available In: Client and Server Applications

Gets or sets the scale of the column. The scale should only be set for floating-point column types and all other column types should have a scale of -1.

> **Note**
> Attempting to change the scale of a column for a dataset that has been opened will result in an exception.

TDataColumn.SortDirection Property

```
property SortDirection: TSortDirection
```

Available In: Client and Server Applications

Specifies whether the column participates in the active sort for the dataset and, if so, which direction the column should be sorted in.

TDataColumn.Text Property

```
property Text: String
```

Available In: Client and Server Applications

Gets or sets the column display text in the active row as a string value. Reading this property triggers the OnGetText event for the column and assigning a value to this property triggers the OnSetText event for the column.

TDataColumn.Clear Method

```
procedure Clear
```

Available In: Client and Server Applications

Use this method to clear the column value in the active row and set the Null property to True. This will also cause the Modified property to be set to True.

TDataColumn.OnGetText Event

```
property OnGetText: TDataColumnTextEvent
```

Available In: Client and Server Applications

This event is triggered when the TDataColumn Text property is read, giving the application a chance to format the resultant text as required. Visual controls that are bound to a column use the Text property to display the data for the column.

TDataColumn.OnSetText Event

```
property OnSetText: TDataColumnTextEvent
```

Available In: Client and Server Applications

This event is triggered when the TDataColumn Text property is assigned a value, giving the application a chance to format or parse the text as required before it is actually stored in the column. Visual controls that are bound to a column use the Text property to assign the data for the column as it is modified by the end user.

10.51 TDataColumns Component

Unit: WebData

Inherits From TCollection

Available In: Client and Server Applications

The TDataColumns class represents the list of columns in a TDataSet component and contains functionality for managing the columns.

Properties	Methods	Events
Column		

TDataColumns.Column Property

```
property Column[Index: Integer]: TDataColumn

property Column[const Name: String]: TDataColumn
```

Available In: Client and Server Applications

Accesses all columns in the dataset by index or by name.

10.52 TDataColumnValue Component

Unit: WebData

Inherits From TObject

Available In: Client and Server Applications

The TDataColumnValue class represents a column value in a TDataSet component and is used by the TDataSet OldValue property to provide access to the old value of a column when the dataset is in the process of being updated.

Properties	Methods	Events
AsBlob		
AsBoolean		
AsDate		
AsDateTime		
AsFloat		
AsInteger		
AsString		
AsTime		

TDataColumnValue.AsBlob Property

```
property AsBlob: TStream
```

Available In: Server Applications

Gets or sets the column value in the active row using a TStream class interface. If the column value cannot be expressed as a BLOB stream, then an exception will be raised.

TDataColumnValue.AsBoolean Property

```
property AsBoolean: Boolean
```

Available In: Client and Server Applications

Gets the old column value in the active row as a boolean value. If the old column value cannot be expressed as a boolean value, then an exception will be raised.

TDataColumnValue.AsDate Property

```
property AsDate: DateTime
```

Available In: Client and Server Applications

Gets the old column value in the active row as a date value. If the old column value cannot be expressed as a date value, then an exception will be raised.

TDataColumnValue.AsDateTime Property

```
property AsDateTime: DateTime
```

Available In: Client and Server Applications

Gets the old column value in the active row as a date/time value. If the old column value cannot be expressed as a date/time value, then an exception will be raised.

TDataColumnValue.AsFloat Property

```
property AsFloat: Double
```

Available In: Client and Server Applications

Gets the old column value in the active row as a floating-point value. If the old column value cannot be expressed as a floating-point value, then an exception will be raised.

TDataColumnValue.AsInteger Property

```
property AsInteger: Integer
```

Available In: Client and Server Applications

Gets the old column value in the active row as an integer value. If the old column value cannot be expressed as an integer value, then an exception will be raised.

TDataColumnValue.AsString Property

```
property AsString: String
```

Available In: Client and Server Applications

Gets the old column value in the active row as a string value. If the old column value cannot be expressed as a string value, then an exception will be raised.

TDataColumnValue.AsTime Property

```
property AsTime: DateTime
```

Available In: Client and Server Applications

Gets the old column value in the active row as a time value. If the old column value cannot be expressed as a time value, then an exception will be raised.

10.53 TDataSet Component

Unit: WebData

Inherits From TComponent

Available In: Client and Server Applications

The TDataSet component represents a dataset and includes functionality for opening, loading, navigating, searching, sorting, updating, and closing datasets. Every dataset that is created is automatically added to the DataSets property for the TDatabase instance passed as the Owner parameter to the TDataSet Create constructor.

Properties	Methods	Events
AutoEdit	BeginControlUpdate	AfterCancel
BaseRowsURL	Cancel	AfterClose
BOF	CheckBrowseMode	AfterDelete
ColumnCount	Close	AfterInsert
Columns	Delete	AfterLoad
DataSetName	DisableControls	AfterLoadColumns
Editing	Empty	AfterOpen
EOF	EnableControls	AfterSave
IncludeInTransaction	EndControlUpdate	AfterScroll
LocalizeDateTimeColumns	Find	AfterUpdate
Modified	First	BeforeCancel
OwnerDatabase	FreeBookmark	BeforeClose
Params	GetColumns	BeforeDelete
RowCount	GetRows	BeforeInsert
RowID	GotoBookmark	BeforeLoad
RowNo	Insert	BeforeLoadColumns
SortCaseInsensitive	Last	BeforeOpen
Sorted	LoadColumns	BeforeSave
SortLocaleInsensitive	LoadRows	BeforeScroll
State	MoveBy	BeforeUpdate
	MoveTo	OnCalculateRow
	Next	OnInitRow
	Open	OnLoadColumnsError
	Prior	OnLoadError
	SaveBookmark	OnRowChanged
	SetControlRowNo	OnSortChanged
	Sort	OnStateChange

	Update	

TDataSet.AutoEdit Property

```
property AutoEdit: Boolean
```

Available In: Client and Server Applications

Specifies whether the dataset can be automatically edited by the user using data-bound visual controls.

TDataSet.BaseRowsURL Property

```
property BaseRowsURL: String
```

Available In: Client Applications

Indicates the base URL that will be used for retrieving the rows for the dataset. This URL is a combination of:

- The OwnerDatabase BaseURL property
- The DataSetName property
- The hard-coded "data" resource

This property is used by bound controls to build the proper URLs for loading BLOB data from the web server.

TDataSet.BOF Property

```
property BOF: Boolean
```

Available In: Client and Server Applications

Indicates whether the row pointer is on the first row in the set of rows in the dataset. This property is only set to True if the First method is called, if the row pointer attempts to navigate beyond the first row using the Prior method, or if the dataset is empty.

TDataSet.ColumnCount Property

```
property ColumnCount: Integer
```

Available In: Client and Server Applications

Indicates the number of columns in the dataset.

TDataSet.Columns Property

```
property Columns: TDataColumns
```

Available In: Client and Server Applications

Provides access to the columns in the dataset.

TDataSet.DataSetName Property

```
property DataSetName: String
```

Available In: Client and Server Applications

Specifies the name of the dataset to use when interacting with the database API.

TDataSet.Editing Property

```
property Editing: Boolean
```

Available In: Client and Server Applications

Indicates whether the dataset is currently inserting or updating the active row via the Insert or Update methods, or if the dataset is currently searching for a row via the Find method.

TDataSet.EOF Property

```
property EOF: Boolean
```

Available In: Client and Server Applications

Indicates whether the row pointer is on the last row in the set of rows in the dataset. This property is only set to True if the Last method is called, the row pointer attempts to navigate beyond the last row using the Next method, or if the dataset is empty.

TDataSet.IncludeInTransaction Property

```
property IncludeInTransaction: Boolean
```

Specifies whether the dataset should be included in any transactions started using the TDatabase StartTransaction method, or indirectly by setting the TDatabase AutoTransactions property to True.

TDataSet.LocalizeDateTimeColumns Property

```
property LocalizeDateTimeColumns: Boolean
```

Available In: Client and Server Applications

Specifies whether the dataset will localize date/time column values when converting them to/from strings. The default value is False.

TDataSet.Modified Property

```
property Modified: Boolean
```

Available In: Client and Server Applications

Indicates whether the active row in the dataset has been modified since the Insert, Update, or Find methods have been called.

TDataSet.OwnerDatabase Property

```
property OwnerDatabase: TDatabase
```

Available In: Client and Server Applications

Indicates the TDatabase instance that owns the dataset.

TDataSet.Params Property

```
property Params: TStrings
```

Available In: Client and Server Applications

Specifies any custom dataset-specific parameters to use with any dataset commands for this dataset. These parameters are included with any relevant database API requests and are useful for specifying additional parameters for the web server to use when executing dataset commands.

TDataSet.RowCount Property

```
property RowCount: Integer
```

Available In: Client and Server Applications

Indicates the number of rows in the dataset.

TDataSet.RowID Property

```
property RowID: Integer
```

Available In: Client and Server Applications

Gets or sets the row ID for the active row in the dataset. If this property is set, then the dataset will automatically navigate to the row that contains the specified row ID, if a row exists with a matching row ID. Each row in a dataset contains an ID that uniquely identifies the row in the dataset.

TDataSet.RowNo Property

```
property RowNo: Integer
```

Available In: Client and Server Applications

Gets or sets the row number for the active row in the dataset. If this property is set, then the dataset will automatically navigate to the specified row number, if the row number is greater than 0 and less than or equal to the RowCount property.

Note
The row number for a row is a **logical** construct, which means that it can change as the dataset is sorted, or as rows are inserted or deleted in the dataset.

TDataSet.SortCaseInsensitive Property

```
property SortCaseInsensitive: Boolean
```

Available In: Client and Server Applications

Specifies whether or not an active sort on the dataset, as indicated by the Sorted property, should be case-sensitive.

Note
Changing this property will trigger a sort on the dataset if the Sorted property is True.

TDataSet.Sorted Property

```
property Sorted: Boolean
```

Available In: Client and Server Applications

Indicates whether any of the Columns in the dataset have their SortDirection property set to sdAscending or sdDescending.

Note
This property does **not** indicate whether the dataset has actually been sorted yet via the Sort method. The sorting is designed this way in order to allow multiple columns to be designated as sort columns without triggering an automatic sort operation each time.

TDataSet.SortLocaleInsensitive Property

```
property SortLocaleInsensitive: Boolean
```

Available In: Client and Server Applications

Specifies whether or not an active sort on the dataset, as indicated by the Sorted property, should be locale-sensitive.

Note
Changing this property will trigger a sort on the dataset if the Sorted property is True.

TDataSet.State Property

```
property State: TDataSetState
```

Available In: Client and Server Applications

Indicates the state of the dataset.

TDataSet.BeginControlUpdate Method

```
procedure BeginControlUpdate
```

Available In: Client and Server Applications

Use this method to disable any notifications to data-bound controls and the triggering of dataset events until the EndControlUpdate method is called. Both of these methods are referenced-counted, so nested calls to the BeginControlUpdate method result in the reference count being incremented, while nested calls to the EndControlUpdate method result in the reference count being decremented. Once the reference count reaches 0, notifications to data-bound controls and the triggering of dataset events will resume.

Note
This method is primarily intended to be used internally by data-bound, multi-row controls when responding to dataset events. The difference between this method and the DisableControls method is that this method also disables the triggering of all dataset events that might cause infinite recursion.

TDataSet.Cancel Method

```
procedure Cancel
```

Available In: Client and Server Applications

Use this method to cancel any active Insert or Update operation, changing the State to dsBrowse.

TDataSet.CheckBrowseMode Method

```
function CheckBrowseMode: Boolean
```

Available In: Client and Server Applications

Checks to see if the dataset State is set to dsBrowse. If the state is dsInsert or dsUpdate and the active row has been modified, then the Save method will be called. If the active row has not been modified, then the Cancel method will be called instead.

TDataSet.Close Method

```
procedure Close
```

Available In: Client and Server Applications

Use this method to close an open dataset.

> **Note**
> Datasets must be closed in order to modify the Columns in the dataset.

TDataSet.Delete Method

```
procedure Delete
```

Available In: Client and Server Applications

Use this method to delete the active row in the dataset. Before the deletion occurs, the BeforeDelete event is triggered, allowing the deletion to be aborted, if necessary. After the deletion occurs, the AfterDelete event is triggered.

Note
If a transaction is active, as indicated by the TDatabase InTransaction property, then the Delete method also logs the current delete operation for inclusion in the current transaction. Please see the Transactions topic for more information.

TDataSet.DisableControls Method

```
procedure DisableControls
```

Available In: Client and Server Applications

Use this method to disable any notifications to data-bound controls until the EnableControls method is called, which is useful when performing lengthy operations on datasets that also have controls bound to them. Both of these methods are referenced-counted, so nested calls to the DisableControls method result in the reference count being incremented, while nested calls to the EnableControls method result in the reference count being decremented. Once the reference count reaches 0, notifications to data-bound controls will resume and the controls will actively reflect any changes to the active row in the dataset.

TDataSet.Empty Method

```
procedure Empty
```

Available In: Client and Server Applications

Use this method to remove all rows from the dataset.

Warning
This method does not log the removal of the rows, even if a transaction is currently active for the global TDatabase instance.

TDataSet.EnableControls Method

```
procedure EnableControls
```

Available In: Client and Server Applications

Use this method to enable any notifications to data-bound controls that were disabled using the DisableControls method. Both of these methods are referenced-counted, so nested calls to the DisableControls method result in the reference count being incremented, while nested calls to the EnableControls method result in the reference count being decremented. Once the reference count reaches 0, notifications to data-bound controls will resume and the controls will actively reflect any changes to the active row in the dataset.

TDataSet.EndControlUpdate Method

```
procedure EndControlUpdate
```

Available In: Client and Server Applications

Use this method to enable any notifications to data-bound controls and the triggering of dataset events that were disabled using the BeginControlUpdate method. Both of these methods are referenced-counted, so nested calls to the BeginControlUpdate method result in the reference count being incremented, while nested calls to the EndControlUpdate method result in the reference count being decremented. Once the reference count reaches 0, notifications to data-bound controls and the triggering of dataset events will resume.

Note
This method is primarily intended to be used internally by data-bound, multi-row controls when responding to dataset events. The difference between this method and the EnableControls method is that this method also disables the triggering of all dataset events that might cause infinite recursion.

TDataSet.Find Method

```
function Find(ColumnsToSearch: array of String; SearchValues:
      array of Variant; NearestMatch: Boolean=False; CaseInsensitive:
      Boolean=False; LocaleInsensitive: Boolean=False): Boolean
```

Available In: Client and Server Applications

Use this method to complete a find operation and perform the actual search of the dataset for a row that matches the column values provided in the ColumnsToSearch and SearchValues parameters. This method will return True if a row is found with the specified column values, or False if not. Specify True for the NearestMatch parameter to cause the search to return the closest match to the desired row, if the desired row cannot be found. Specify True for the CaseInsensitive parameter to cause the search to be executed in a case-insensitive manner for any string columns. Specify True for the LocaleInsensitive parameter to cause the search to be executed in a locale-insensitive manner for any string columns.

Note
The NearestMatch parameter can only be set to True if there is an active sort on the dataset, as specified by the Sorted property, and the ColumnsToSearch parameter matchs the columns in the active sort. Also, the CaseInsensitive and LocaleInsensitive parameters must also match the SortCaseInsensitive and SortLocaleInsensitive properties if there is an active sort on the dataset.

TDataSet.First Method

```
procedure First
```

Available In: Client and Server Applications

Use this method to move to the first row in the dataset, taking into account any active sort on the dataset.

TDataSet.FreeBookmark Method

```
procedure FreeBookmark
```

Available In: Client and Server Applications

Use this method to free the bookmark last saved via the SaveBookmark method. Bookmark operations are stack-based, so every call to the SaveBookmark method must be followed by a call to the GotoBookmark or FreeBookmark method in order to ensure proper operation.

TDataSet.GetColumns Method

```
function GetColumns: String
```

Available In: Client and Server Applications

Use this method to retrieve the defined Columns for the dataset as a JSON string that can either be stored using a TPersistentStorage instance, or sent to a web server using a TServerRequest instance.

TDataSet.GetRows Method

```
function GetRows: String
```

Available In: Client and Server Applications

Use this method to retrieve the rows in the dataset as a JSON string that can either be stored using a TPersistentStorage instance, or sent to a web server using a TServerRequest instance.

TDataSet.GotoBookmark Method

```
function GotoBookmark: Boolean
```

Available In: Client and Server Applications

Use this method to move the row pointer to the bookmark last saved via the SaveBookmark method, and then free the bookmark. Bookmark operations are stack-based, so every call to the SaveBookmark method must be followed by a call to the GotoBookmark or FreeBookmark method in order to ensure proper operation.

TDataSet.Insert Method

```
procedure Insert(Append: Boolean=False)
```

Available In: Client and Server Applications

Use this method to begin inserting a new row in the dataset. If the Append parameter is True, then the new row will be appended to the end of the dataset, and if the Append parameter is False, then the new row will be inserted at the active row position in the dataset. Before the insert occurs, the BeforeInsert event is triggered, allowing the insertion to be aborted, if necessary. The OnInitRow event is then triggered to allow the new row to be initialized without flagging the row as being modified. After the insertion occurs, the AfterInsert event is triggered.

Use the Save method to complete the insertion of the new row.

TDataSet.Last Method

```
procedure Last
```

Available In: Client and Server Applications

Use this method to move to the last row in the dataset, taking into account any active sort on the dataset.

TDataSet.LoadColumns Method

```
procedure LoadColumns

procedure LoadColumns(const ColumnData: String)
```

Available In: Client and Server Applications

Use this method to load the columns for a dataset.

For client applications, the columns are loaded from a JSON string parameter. Please see the Web Server Administration API - Databases topic for more information on how the JSON string should be formatted.

For server applications, the columns are loaded directly from the web server when this method is called, and no JSON string parameter is required.

Note
The dataset must be closed or an exception will be raised when this method is called.

TDataSet.LoadRows Method

```
procedure LoadRows(Append: Boolean=False)

procedure LoadRows(const RowData: String; Append: Boolean=False)
```

Available In: Client and Server Applications

Use this method to load the rows for a dataset. The Append parameter determines if the rows should be appended to the existing dataset rows, or if the dataset should be emptied before the rows are loaded.

For client applications, the rows are loaded from a JSON string parameter. Please see the Web Server Administration API - Databases topic for more information on how the JSON string should be formatted.

For server applications, the rows are loaded directly from the web server when this method is called, and no JSON string parameter is required.

Note
The dataset must be open or an exception will be raised when this method is called.

TDataSet.MoveBy Method

```
function MoveBy(Value: Integer): Integer
```

Available In: Client and Server Applications

Use this method to move N number of rows forward or backward relative to the active row in the dataset, taking into account any active sort on the dataset. If the number of rows passed as the parameter to this method is negative, then the row pointer is moved backward in the dataset. If the number of rows is positive, then the row pointer is moved forward in the dataset.

If the row pointer cannot be moved the specified number of rows without proceeding past the first or last rows in the dataset, then the row pointer will be positioned at the first or last row in the dataset, setting the BOF and/or EOF properties as required.

TDataSet.MoveTo Method

```
procedure MoveTo(Value: Integer)
```

Available In: Client and Server Applications

Use this method to move to a specific position in the dataset, taking into account any active sort on the dataset. If the position specified is less than 1 or greater than the number of rows in the dataset, then the row pointer will be positioned at the first or last row in the dataset, setting the BOF and/or EOF properties as required.

TDataSet.Next Method

```
procedure Next
```

Available In: Client and Server Applications

Use this method to move the row pointer to the next row in the dataset, taking into account any active sort on the dataset. If the row pointer is already pointing to the last row in the dataset, then the EOF property will be set to True.

TDataSet.Open Method

```
procedure Open
```

Available In: Client and Server Applications

Use this method to open a closed dataset.

Note
Datasets must be open in order to load rows into the dataset via the LoadRows method.

TDataSet.Prior Method

```
procedure Prior
```

Available In: Client and Server Applications

Use this method to move the row pointer to the prior row in the dataset, taking into account any active sort on the dataset. If the row pointer is already pointing to the first row in the dataset, then the BOF property will be set to True.

TDataSet.SaveBookmark Method

```
procedure SaveBookmark
```

Available In: Client and Server Applications

Use this method to save the row pointer as a bookmark. Bookmark operations are stack-based, so every call to the SaveBookmark method must be followed by a call to the GotoBookmark or FreeBookmark method in order to ensure proper operation.

TDataSet.SetControlRowNo Method

```
procedure SetControlRowNo(ARowNo: Integer)
```

Available In: Client and Server Applications

Sets the row number for the active row in the dataset without firing any dataset events.

> **Note**
> This method is primarily intended to be used by data-bound, multi-row controls to update the row number for the dataset without any validation of the row number or firing of any dataset events that might cause infinite recursion.

TDataSet.Sort Method

```
procedure Sort
```

Available In: Client and Server Applications

Use this method to sort the dataset when the Sorted property is True and sort columns have been specified for the dataset. The TDataColumn SortDirection property setting controls which columns are sorted, and how.

If no columns have a SortDirection property other than sdNone, then this method does nothing.

The SortCaseInsensitive and SortLocaleInsensitive properties control how the sort is performed.

Note
The Sort method only needs to be called once after the sort directions are initially assigned for the columns. After that point, any row operations will automatically maintain the active sort.

TDataSet.Update Method

```
procedure Update
```

Available In: Client and Server Applications

Use this method to begin updating the active row in the dataset. Before the update occurs, the BeforeUpdate event is triggered, allowing the update to be aborted, if necessary. After the update occurs, the AfterUpdate event is triggered.

Use the Save method to complete the update of the active row.

TDataSet.AfterCancel Event

```
property AfterCancel: TNotifyEvent
```

Available In: Client and Server Applications

This event is triggered after the Cancel method completes.

TDataSet.AfterClose Event

```
property AfterClose: TNotifyEvent
```

Available In: Client and Server Applications

This event is triggered after the Close method completes.

TDataSet.AfterDelete Event

```
property AfterDelete: TNotifyEvent
```

Available In: Client and Server Applications

This event is triggered after the Delete method completes.

TDataSet.AfterInsert Event

```
property AfterInsert: TNotifyEvent
```

Available In: Client and Server Applications

This event is triggered after the Insert method completes.

TDataSet.AfterLoad Event

```
property AfterLoad: TNotifyEvent
```

Available In: Client and Server Applications

This event is triggered after the LoadRows method completes.

TDataSet.AfterLoadColumns Event

```
property AfterLoadColumns: TNotifyEvent
```

Available In: Client and Server Applications

This event is triggered after the LoadColumns method completes.

TDataSet.AfterOpen Event

```
property AfterOpen: TNotifyEvent
```

Available In: Client and Server Applications

This event is triggered after the Open method completes.

TDataSet.AfterSave Event

```
property AfterSave: TNotifyEvent
```

Available In: Client and Server Applications

This event is triggered after the Save method completes.

TDataSet.AfterScroll Event

```
property AfterScroll: TNotifyEvent
```

Available In: Client and Server Applications

This event is triggered after any navigation operation completes for a dataset. The following TDataSet methods cause the AfterScroll event to be triggered:

First
Prior
Next
Last
MoveBy
MoveTo
Find
Sort
GotoBookmark

TDataSet.AfterUpdate Event

```
property AfterUpdate: TNotifyEvent
```

Available In: Client and Server Applications

This event is triggered after the Update method completes.

TDataSet.BeforeCancel Event

```
property BeforeCancel: TDataSetEvent
```

Available In: Client and Server Applications

This event is triggered before the Cancel method starts. Return False from the event handler to prevent the cancel from occurring.

TDataSet.BeforeClose Event

```
property BeforeClose: TDataSetEvent
```

Available In: Client and Server Applications

This event is triggered before the Close method starts. Return False from the event handler to prevent the close from occurring.

TDataSet.BeforeDelete Event

```
property BeforeDelete: TDataSetEvent
```

Available In: Client and Server Applications

This event is triggered before the Delete method starts. Return False from the event handler to prevent the deletion from occurring.

TDataSet.BeforeInsert Event

```
property BeforeInsert: TDataSetEvent
```

Available In: Client and Server Applications

This event is triggered before the Insert method starts. Return False from the event handler to prevent the insertion from occurring.

TDataSet.BeforeLoad Event

```
property BeforeLoad: TDataSetEvent
```

Available In: Client and Server Applications

This event is triggered before the LoadRows method starts. Return False from the event handler to prevent the rows load operation from occurring.

TDataSet.BeforeLoadColumns Event

```
property BeforeLoadColumns: TDataSetEvent
```

Available In: Client and Server Applications

This event is triggered before the LoadColumns method starts. Return False from the event handler to prevent the columns load operation from occurring.

TDataSet.BeforeOpen Event

```
property BeforeOpen: TDataSetEvent
```

Available In: Client and Server Applications

This event is triggered before the Open method starts. Return False from the event handler to prevent the open from occurring.

TDataSet.BeforeSave Event

```
property BeforeSave: TDataSetEvent
```

Available In: Client and Server Applications

This event is triggered before the Save method starts. Return False from the event handler to prevent the save from occurring.

TDataSet.BeforeScroll Event

```
property BeforeScroll: TDataSetEvent
```

Available In: Client and Server Applications

This event is triggered before any navigation operation starts for a dataset. The following TDataSet methods cause the BeforeScroll event to be triggered:

First
Prior
Next
Last
MoveBy
MoveTo
Find
Sort
GotoBookmark

Return False from the event handler to prevent the navigation from occurring.

TDataSet.BeforeUpdate Event

```
property BeforeUpdate: TDataSetEvent
```

Available In: Client and Server Applications

This event is triggered before the Update method starts. Return False from the event handler to prevent the update from occurring.

TDataSet.OnCalculateRow Event

```
property OnCalculateRow: TDataRowEvent
```

Available In: Client and Server Applications

This event is triggered whenever a column in a row of dataset is updated. You can use this event to re-compute any calculated columns defined in the dataset.

> **Note**
> You cannot modify non-calculated columns in an event handler attached to this event.

TDataSet.OnInitRow Event

```
property OnInitRow: TNotifyEvent
```

Available In: Client and Server Applications

This event is triggered when the Insert method is called and provides an opportunity to initialize a new row with default values. Any columns modified in this event handler have their Modified property reset after the event handler for this event is done executing.

TDataSet.OnLoadColumnsError Event

```
property OnLoadColumnsError: TDataSetErrorEvent
```

Available In: Client and Server Applications

This event is triggered whenever an error occurs during the execution of the LoadColumns method.

Note
If an event handler is not assigned to this event, then the error will be raised as an exception.

TDataSet.OnLoadError Event

```
property OnLoadError: TDataSetErrorEvent
```

Available In: Client and Server Applications

This event is triggered whenever an error occurs during the execution of the LoadRows method.

Note
If an event handler is not assigned to this event, then the error will be raised as an exception.

TDataSet.OnRowChanged Event

```
property OnRowChanged: TDataRowEvent
```

Available In: Client and Server Applications

This event is triggered whenever the active row, or any column in the active row, is changed.

TDataSet.OnSortChanged Event

```
property OnSortChanged: TDataRowEvent
```

Available In: Client and Server Applications

This event is triggered whenever the active sort is changed.

TDataSet.OnStateChange Event

```
property OnStateChange: TNotifyEvent
```

Available In: Client and Server Applications

This event is triggered whenever the State property changes.

10.54 TDataSetController Component

Unit: WebData

Inherits From TComponent

Available In: Visual Client Applications

The TDataSetController component provides a way for a developer to respond to changes in a TDataSet instance that is located on the same form/database or another form/database in a similar fashion to data-bound controls. It also allows the developer to initiate the editing of a TDataSet instance via the Edit method. The TDataSet instance to attach to is specified via the DataSet property.

Properties	Methods	Events
DataSet	Edit	OnDataChanged
		OnSortChanged
		OnStateChange

TDataSetController.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset that the controller should attach to.

TDataSetController.Edit Method

```
function Edit: Boolean
```

Available In: Visual Client Applications

Use this method to cause the attached DataSet to begin editing the current row. If the dataset is empty, then a new row will be added.

TDataSetController.OnDataChanged Event

```
property OnDataChanged: TDataRowEvent
```

Available In: Visual Client Applications

This event is triggered whenever the DataSet is changed via scrolling and navigation, inserts, updates, or deletes.

TDataSetController.OnSortChanged Event

```
property OnSortChanged: TDataRowEvent
```

Available In: Visual Client Applications

This event is triggered whenever the DataSet's active sort changes.

TDataSetController.OnStateChange Event

```
property OnStateChange: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the DataSet's State changes.

10.55 TDataSetToolBar Component

Unit: WebTlbrs

Inherits From TToolBarControl

Available In: Visual Client Applications

The TDataSetToolBar class represents a dataset toolbar control. When a dataset toolbar control is bound to a TDataSet instance, it can be used to navigate and update the dataset and its buttons will automatically be enabled/disabled to reflect the current dataset state.

Properties	Methods	Events
Background		OnAnimationComplete
Border		OnAnimationsComplete
Buttons		OnButtonClick
Corners		OnHide
Cursor		OnMove
DataSet		OnShow
Hint		OnSize
InsetShadow		
Opacity		

TDataSetToolBar.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

TDataSetToolBar.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TDataSetToolBar.Buttons Property

```
property Buttons: TDataSetToolBarButtons
```

Available In: Visual Client Applications

Contains the pre-defined buttons for the toolbar.

TDataSetToolBar.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TDataSetToolBar.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TDataSetToolBar.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

TDataSetToolBar.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TDataSetToolBar.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Available In: Visual Client Applications

Specifies the inset shadow for the control.

TDataSetToolBar.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TDataSetToolBar.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TDataSetToolBar.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TDataSetToolBar.OnButtonClick Event

```
property OnButtonClick: TClickEvent
```

Available In: Visual Client Applications

This event is triggered whenever one of the toolbar buttons is clicked.

Return True to allow the default click behavior and False to prevent the default click behavior from occurring.

TDataSetToolBar.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TDataSetToolBar.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TDataSetToolBar.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TDataSetToolBar.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

10.56 TDataSetToolBarButton Component

Unit: WebTlbrs

Inherits From TToolBarButton

Available In: Visual Client Applications

The TDataSetToolBarButton class represents a dataset toolbar button control contained within a TDataSetToolBar instance.

Properties	Methods	Events

10.57 TDataSetToolBarButtons Component

Unit: WebTlbrs

Inherits From TComponent

Available In: Visual Client Applications

The TDataSetToolBarButtons class represents a collection of dataset toolbar button controls for the various operations required for a TDataSet instance bound to the container TDataSetToolBar instance. These buttons can be used to navigate and update the dataset and they will automatically be enabled/disabled to reflect the current dataset state.

Properties	Methods	Events
AppendButton		
CancelButton		
DeleteButton		
FindButton		
FirstButton		
LastButton		
NextButton		
PriorButton		
SaveButton		
UpdateButton		

TDataSetToolBarButtons.AppendButton Property

```
property AppendButton: TToolBarButton
```

Available In: Visual Client Applications

Specifies the properties of the Append button.

TDataSetToolBarButtons.CancelButton Property

```
property CancelButton: TToolBarButton
```

Available In: Visual Client Applications

Specifies the properties of the Cancel button.

TDataSetToolBarButtons.DeleteButton Property

```
property DeleteButton: TToolBarButton
```

Available In: Visual Client Applications

Specifies the properties of the Delete button.

TDataSetToolBarButtons.FindButton Property

```
property FindButton: TToolBarButton
```

Available In: Visual Client Applications

Specifies the properties of the Find button.

Note
By default, the Find button doesn't actually perform any action when clicked. You need to assign an event handler to the TDataSetToolBar OnButtonClick event in order to perform an action when the button is clicked.

TDataSetToolBarButtons.FirstButton Property

```
property FirstButton: TToolBarButton
```

Available In: Visual Client Applications

Specifies the properties of the First button.

TDataSetToolBarButtons.LastButton Property

```
property LastButton: TToolBarButton
```

Available In: Visual Client Applications

Specifies the properties of the Last button.

TDataSetToolBarButtons.NextButton Property

```
property NextButton: TToolBarButton
```

Available In: Visual Client Applications

Specifies the properties of the Next button.

TDataSetToolBarButtons.PriorButton Property

```
property PriorButton: TToolBarButton
```

Available In: Visual Client Applications

Specifies the properties of the Prior button.

TDataSetToolBarButtons.SaveButton Property

```
property SaveButton: TToolBarButton
```

Available In: Visual Client Applications

Specifies the properties of the Save button.

TDataSetToolBarButtons.UpdateButton Property

```
property UpdateButton: TToolBarButton
```

Available In: Visual Client Applications

Specifies the properties of the Update button.

10.58 TDateEditComboBox Component

Unit: WebEdits

Inherits From TEditComboControl

Available In: Visual Client Applications

The TDateEditComboBox component represents a date edit combo box control. A date edit combo box is a combo box control that allows the user to directly enter a date value or select a date value from a drop-down calendar.

Properties	Methods	Events
Alignment		OnAnimationComplete
AutoComplete		OnAnimationsComplete
CalendarDefaultView		OnButtonClick
CalendarHeight		OnCaptureEnd
CalendarWidth		OnCaptureStart
Cursor		OnCapturing
DataColumn		OnChange
DataSet		OnClick
Direction		OnContextMenu
DropDownPosition		OnDblClick
DropDownVisible		OnDropDownHide
Enabled		OnDropDownShow
Font		OnEnter
Hint		OnExit
MaxLength		OnHide
ReadOnly		OnKeyDown
SelectedDate		OnKeyPress
TabOrder		OnKeyUp
TabStop		OnMouseDown
Text		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd

		OnTouchMove
		OnTouchStart

TDateEditComboBox.Alignment Property

```
property Alignment: TContentAlignment
```

Available In: Visual Client Applications

Specifies the alignment of the input value for the control.

TDateEditComboBox.AutoComplete Property

```
property AutoComplete: TAutoCompleteType
```

Available In: Visual Client Applications

Specifies how to handle auto-completion for the control. Auto-completion allows the browser to display a list of suggestions for input values, based upon earlier input values entered by the user.

TDateEditComboBox.CalendarDefaultView Property

```
property CalendarDefaultView: TCalendarView
```

Available In: Visual Client Applications

Specifies the default view for the drop-down calendar. The default view determines both the initial view shown in the calendar after it is created, as well as the minimum view that the user is permitted to navigate to. The default value is cvMonth.

TDateEditComboBox.CalendarHeight Property

```
property CalendarHeight: Integer
```

Available In: Visual Client Applications

Specifies the height of the drop-down calendar.

TDateEditComboBox.CalendarWidth Property

```
property CalendarWidth: Integer
```

Available In: Visual Client Applications

Specifies the width of the drop-down calendar.

TDateEditComboBox.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TDateEditComboBox.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TDateEditComboBox.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

TDateEditComboBox.Direction Property

```
property Direction: TContentDirection
```

Available In: Visual Client Applications

Specifies the direction in which the text is displayed/edited.

TDateEditComboBox.DropDownPosition Property

```
property DropDownPosition: TDropDownPosition
```

Available In: Visual Client Applications

Specifies where the drop-down calendar will be positioned.

TDateEditComboBox.DropDownVisible Property

```
property DropDownVisible: Boolean
```

Available In: Visual Client Applications

Indicates whether the drop-down calendar is visible.

TDateEditComboBox.Enabled Property

```
property Enabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TDateEditComboBox.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TDateEditComboBox.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TDateEditComboBox.MaxLength Property

```
property MaxLength: Integer
```

Available In: Visual Client Applications

Specifies the maximum allowable length, in characters, of the Text property for the control. A value of 0 specifies an unlimited allowable length.

TDateEditComboBox.ReadOnly Property

```
property ReadOnly: Boolean
```

Available In: Visual Client Applications

Specifies whether the control's input value can be modified by the user. The default value is False.

> **Note**
> The input value can always be programmatically modified.

TDateEditComboBox.SelectedDate Property

```
property SelectedDate: DateTime
```

Available In: Visual Client Applications

Specifies the selected date for the control.

TDateEditComboBox.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TDateEditComboBox.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TDateEditComboBox.Text Property

```
property Text: String
```

Available In: Visual Client Applications

Specifies the control's input value as a string.

TDateEditComboBox.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TDateEditComboBox.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TDateEditComboBox.OnButtonClick Event

```
property OnButtonClick: TClickEvent
```

Available In: Visual Client Applications

This event is triggered whenever the associated combo button is clicked.

Return True to allow the default click behavior and False to prevent the default click behavior from occurring.

TDateEditComboBox.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TDateEditComboBox.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TDateEditComboBox.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TDateEditComboBox.OnChange Event

```
property OnChange: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TDateEditComboBox.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TDateEditComboBox.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TDateEditComboBox.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TDateEditComboBox.OnDropDownHide Event

```
property OnDropDownHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the associated drop-down control is hidden.

TDateEditComboBox.OnDropDownShow Event

```
property OnDropDownShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the associated drop-down control is shown.

TDateEditComboBox.OnEnter Event

```
property OnEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control obtains input focus.

TDateEditComboBox.OnExit Event

```
property OnExit: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control loses input focus.

TDateEditComboBox.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TDateEditComboBox.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses a key or key combination.

TDateEditComboBox.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TDateEditComboBox.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user releases a key or key combination.

TDateEditComboBox.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TDateEditComboBox.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TDateEditComboBox.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TDateEditComboBox.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TDateEditComboBox.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TDateEditComboBox.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TDateEditComboBox.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TDateEditComboBox.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TDateEditComboBox.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TDateEditComboBox.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TDateEditComboBox.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TDateEditComboBox.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.59 TDialog Component

Unit: WebForms

Inherits From TDialogControl

Available In: Visual Client Applications

The TDialog component represents a dialog form control with a border and a caption bar with a close button. TDialogButton components can be used with a TDialog instance to provide standard dialog keystroke and button click functionality.

Properties	Methods	Events
ActivateOnClick		OnAnimationComplete
CaptionBar		OnAnimationsComplete
Client		OnCaptionBarDblClick
CloseOnEscape		OnCaptureEnd
Corners		OnCaptureStart
Cursor		OnCapturing
Opacity		OnClick
OutsetShadow		OnClose
Sizable		OnCloseQuery
		OnContextMenu
		OnDblClick
		OnHide
		OnKeyDown
		OnKeyPress
		OnKeyUp
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMouseWheel
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove

		OnTouchStart

TDialog.ActivateOnClick Property

```
property ActivateOnClick: Boolean
```

Available In: Visual Client Applications

Specifies whether the dialog should automatically be brought to the front when it, or any child controls, are clicked.

Note
This property only has an effect when the dialog is parented to another control. By default, dialogs always are brought to the front when clicked.

TDialog.CaptionBar Property

```
property CaptionBar: TDialogCaptionBar
```

Available In: Visual Client Applications

Specifies the properties of the caption bar for the control.

TDialog.Client Property

```
property Client: TDialogClient
```

Available In: Visual Client Applications

Specifies the properties of the client area for the control.

TDialog.CloseOnEscape Property

```
property CloseOnEscape: Boolean
```

Available In: Visual Client Applications

Specifies whether the dialog is automatically closed when the user hits the Escape key. The default value is True.

Note
If there is a TDialogButton instance on the dialog with its ModalCancel property set to True, then the button will be clicked when the escape key is pressed and this property will ignored.

TDialog.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TDialog.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TDialog.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TDialog.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Available In: Visual Client Applications

Specifies the outset shadow for the control.

TDialog.Sizable Property

```
property Sizable: Boolean
```

Available In: Visual Client Applications

Specifies whether the dialog should be sizable. When a dialog is sizable, the user will be able to click on the bottom-right border of the dialog to begin sizing the dialog interactively, and moving the mouse while holding down the left mouse button will allow the user to size the dialog according to their needs.

TDialog.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TDialog.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TDialog.OnCaptionBarDblClick Event

```
property OnCaptionBarDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the caption bar is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TDialog.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TDialog.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TDialog.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TDialog.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TDialog.OnClose Event

```
property OnClose: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the dialog is closed by the user via the caption bar close button, or when the Close method is called.

TDialog.OnCloseQuery Event

```
property OnCloseQuery: TCloseQueryEvent
```

Available In: Visual Client Applications

This event is triggered when the dialog is closed by the user via the caption bar close button, or when the Close method is called.

Return True to allow the close to continue, or False to prevent the dialog from closing.

TDialog.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TDialog.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TDialog.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TDialog.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and the user presses a key or key combination.

TDialog.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and presses/releases a key or key combination.

TDialog.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and the user releases a key or key combination.

TDialog.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TDialog.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TDialog.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TDialog.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TDialog.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TDialog.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

Available In: Visual Client Applications

This event is triggered whenever the mouse wheel is rotated forward or backward.

TDialog.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TDialog.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TDialog.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TDialog.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TDialog.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TDialog.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TDialog.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.60 TDialogButton Component

Unit: WebBtns

Inherits From TButton

Available In: Visual Client Applications

The TDialogButton component represents a dialog button control. A dialog button control allows the user to initiate a specific action by using a mouse click or by pushing the spacebar or enter key. In addition, the ModalResult property can be used when a dialog button control is placed on a modal form. If the ModalResult property is set, then clicking the dialog button control will automatically cause the form to close and return the same modal result in the ModalResult property for the form.

Properties	Methods	Events
ModalCancel		
ModalDefault		
ModalResult		

TDialogButton.ModalCancel Property

```
property ModalCancel: Boolean
```

Available In: Visual Client Applications

Specifies that this button is the cancel button for the parent dialog. If the escape key is pressed while the dialog is active, then the ModalResult property for the dialog will be set to the same value as the ModalResult property of the button.

TDialogButton.ModalDefault Property

```
property ModalDefault: Boolean
```

Available In: Visual Client Applications

Specifies that this button is the default button for the parent dialog. If the enter key is pressed while the dialog is active, then the ModalResult property for the dialog will be set to the same value as the ModalResult property of the button.

TDialogButton.ModalResult Property

```
property ModalResult: TModalResult
```

Available In: Visual Client Applications

Specifies the result to assign to a form's ModalResult property when the button is clicked.

10.61 TDialogCaptionBar Component

Unit: WebForms

Inherits From TCaptionBarControl

Available In: Visual Client Applications

The TDialogCaptionBar component represents the caption bar for a TDialog form control, and includes a caption and a close button.

Properties	Methods	Events
Alignment		
AllowClose		
AllowMove		
Background		
Caption		
Cursor		
Font		
Icon		
Padding		

TDialogCaptionBar.Alignment Property

```
property Alignment: TContentAlignment
```

Available In: Visual Client Applications

Specifies the alignment of the caption in the caption bar.

TDialogCaptionBar.AllowClose Property

```
property AllowClose: Boolean
```

Available In: Visual Client Applications

Specifies whether the close button should be shown in the caption bar.

TDialogCaptionBar.AllowMove Property

```
property AllowMove: Boolean
```

Available In: Visual Client Applications

Specifies whether the user can press and hold a mouse or touch on the caption bar and drag the container dialog to a new position.

TDialogCaptionBar.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

TDialogCaptionBar.Caption Property

```
property Caption: String
```

Available In: Visual Client Applications

Specifies the caption to display in the caption bar.

TDialogCaptionBar.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TDialogCaptionBar.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TDialogCaptionBar.Icon Property

```
property Icon: TIconProperties
```

Available In: Visual Client Applications

Specifies the properties of the icon used with the caption bar.

TDialogCaptionBar.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

10.62 TDialogClient Component

Unit: WebForms

Inherits From TComponent

Available In: Visual Client Applications

The TDialogClient component represents the client area for a TDialog form control.

Properties	Methods	Events
Background		
InsetShadow		
Padding		

TDialogClient.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background for the control.

TDialogClient.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Available In: Visual Client Applications

Specifies the inset shadow for the control.

TDialogClient.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

10.63 TDialogControl Component

Unit: WebForms

Inherits From TFormControl

Available In: Visual Client Applications

The TDialogControl control is the base class for dialogs, and contains all of the core dialog functionality in the form of public methods and protected properties/events that descendant classes can use to create customized dialogs.

Properties	Methods	Events

10.64 TDialogEditComboBox Component

Unit: WebEdits

Inherits From TEditComboControl

Available In: Visual Client Applications

The TDialogEditComboBox component represents a dialog edit combo box control. A dialog edit combo box is a combo box control that allows the user to directly enter an input value or trigger the display of a custom selection dialog by intercepting the combo button clicks.

Properties	Methods	Events
Alignment		OnAnimationComplete
AutoComplete		OnAnimationsComplete
Cursor		OnButtonClick
DataColumn		OnCaptureEnd
DataSet		OnCaptureStart
Direction		OnCapturing
Enabled		OnChange
Font		OnClick
Hint		OnContextMenu
Icon		OnDblClick
MaxLength		OnEnter
ReadOnly		OnExit
SpellCheck		OnHide
TabOrder		OnKeyDown
TabStop		OnKeyPress
Text		OnKeyUp
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove

		OnTouchStart

TDialogEditComboBox.Alignment Property

```
property Alignment: TContentAlignment
```

Available In: Visual Client Applications

Specifies the alignment of the input value for the control.

TDialogEditComboBox.AutoComplete Property

```
property AutoComplete: TAutoCompleteType
```

Available In: Visual Client Applications

Specifies how to handle auto-completion for the control. Auto-completion allows the browser to display a list of suggestions for input values, based upon earlier input values entered by the user.

TDialogEditComboBox.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TDialogEditComboBox.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TDialogEditComboBox.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

TDialogEditComboBox.Direction Property

```
property Direction: TContentDirection
```

Available In: Visual Client Applications

Specifies the direction in which the text is displayed/edited.

TDialogEditComboBox.Enabled Property

```
property Enabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TDialogEditComboBox.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TDialogEditComboBox.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TDialogEditComboBox.Icon Property

```
property Icon: TIconProperties
```

Available In: Visual Client Applications

Specifies the properties of the icon used with the button in the control.

TDialogEditComboBox.MaxLength Property

```
property MaxLength: Integer
```

Available In: Visual Client Applications

Specifies the maximum allowable length, in characters, of the Text property for the control. A value of 0 specifies an unlimited allowable length.

TDialogEditComboBox.ReadOnly Property

```
property ReadOnly: Boolean
```

Available In: Visual Client Applications

Specifies whether the control's input value can be modified by the user. The default value is False.

> **Note**
> The input value can always be programmatically modified.

TDialogEditComboBox.SpellCheck Property

```
property SpellCheck: Boolean
```

Available In: Visual Client Applications

Specifies whether spell-checking will be enabled for the control.

TDialogEditComboBox.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TDialogEditComboBox.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TDialogEditComboBox.Text Property

```
property Text: String
```

Available In: Visual Client Applications

Specifies the control's input value as a string.

TDialogEditComboBox.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TDialogEditComboBox.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TDialogEditComboBox.OnButtonClick Event

```
property OnButtonClick: TClickEvent
```

Available In: Visual Client Applications

This event is triggered whenever the associated combo button is clicked.

Return True to allow the default click behavior and False to prevent the default click behavior from occurring.

TDialogEditComboBox.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TDialogEditComboBox.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TDialogEditComboBox.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TDialogEditComboBox.OnChange Event

```
property OnChange: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TDialogEditComboBox.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TDialogEditComboBox.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TDialogEditComboBox.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TDialogEditComboBox.OnEnter Event

```
property OnEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control obtains input focus.

TDialogEditComboBox.OnExit Event

```
property OnExit: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control loses input focus.

TDialogEditComboBox.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TDialogEditComboBox.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses a key or key combination.

TDialogEditComboBox.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TDialogEditComboBox.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user releases a key or key combination.

TDialogEditComboBox.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TDialogEditComboBox.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TDialogEditComboBox.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TDialogEditComboBox.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TDialogEditComboBox.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TDialogEditComboBox.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TDialogEditComboBox.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TDialogEditComboBox.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TDialogEditComboBox.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TDialogEditComboBox.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TDialogEditComboBox.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TDialogEditComboBox.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.65 TDivElement Component

Unit: WebUI

Inherits From TElement

Available In: Visual Client Applications

The TDivElement class is the default element class for UI elements, and contains all of the element functionality in the form of public methods and properties/events that control classes can use to create any type of control.

> **Note**
> The TDivElement is just a typecast of the TElement class, and is only used for associating an element class type with a specific type of run-time HTML tag ( ).

Properties	Methods	Events

10.66 TDropDownButtonControl Component

Unit: WebEdits

Inherits From TButtonInputControl

Available In: Visual Client Applications

The TDropDownButtonControl control is the base class for drop-down button controls, and contains all of the core drop-down functionality in the form of public methods and protected properties/events that descendant classes can use to create customized drop-down button controls.

Properties	Methods	Events
	HideDropDown	
	ShowDropDown	

TDropDownButtonControl.HideDropDown Method

```
procedure HideDropDown
```

Available In: Visual Client Applications

Use this method to hide the drop-down control associated with the control (if visible).

TDropDownButtonControl.ShowDropDown Method

```
procedure ShowDropDown
```

Available In: Visual Client Applications

Use this method to show the drop-down control associated with the control (if not already visible).

10.67 TDropDownEditControl Component

Unit: WebEdits

Inherits From TEditControl

Available In: Visual Client Applications

The TDropDownEditControl control is the base class for drop-down edit controls, and contains all of the core drop-down functionality in the form of public methods and protected properties/events that descendant classes can use to create customized drop-down edit controls.

Properties	Methods	Events
	HideDropDown	
	ShowDropDown	

TDropDownEditControl.HideDropDown Method

```
procedure HideDropDown
```

Available In: Visual Client Applications

Use this method to hide the drop-down control associated with the control (if visible).

TDropDownEditControl.ShowDropDown Method

```
procedure ShowDropDown
```

Available In: Visual Client Applications

Use this method to show the drop-down control associated with the control (if not already visible).

10.68 TEdit Component

Unit: WebEdits

Inherits From TEditControl

Available In: Visual Client Applications

The TEdit component represents an edit control. An edit control allows the user to directly enter an input value using the keyboard.

Properties	Methods	Events
Alignment		OnAnimationComplete
AutoComplete		OnAnimationsComplete
Cursor		OnCaptureEnd
DataColumn		OnCaptureStart
DataSet		OnCapturing
Direction		OnChange
Enabled		OnClick
Font		OnContextMenu
Hint		OnDblClick
InputType		OnEnter
MaxLength		OnExit
ReadOnly		OnHide
SpellCheck		OnKeyDown
TabOrder		OnKeyPress
TabStop		OnKeyUp
Text		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TEdit.Alignment Property

```
property Alignment: TContentAlignment
```

Available In: Visual Client Applications

Specifies the alignment of the input value for the control.

TEdit.AutoComplete Property

```
property AutoComplete: TAutoCompleteType
```

Available In: Visual Client Applications

Specifies how to handle auto-completion for the control. Auto-completion allows the browser to display a list of suggestions for input values, based upon earlier input values entered by the user.

TEdit.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TEdit.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TEdit.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

TEdit.Direction Property

```
property Direction: TContentDirection
```

Available In: Visual Client Applications

Specifies the direction in which the text is displayed/edited.

TEdit.Enabled Property

```
property Enabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TEdit.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TEdit.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TEdit.InputType Property

```
property InputType: TTextInputType
```

Available In: Visual Client Applications

Specifies the type of text being input into the element by the user. This information is used by the browser to determine how to display and edit the text. For example, in touch environments, this property is used to determine which soft keyboard to display to the user.

TEdit.MaxLength Property

```
property MaxLength: Integer
```

Available In: Visual Client Applications

Specifies the maximum allowable length, in characters, of the Text property for the control. A value of 0 specifies an unlimited allowable length.

TEdit.ReadOnly Property

```
property ReadOnly: Boolean
```

Available In: Visual Client Applications

Specifies whether the control's input value can be modified by the user. The default value is False.

> **Note**
> The input value can always be programmatically modified.

TEdit.SpellCheck Property

```
property SpellCheck: Boolean
```

Available In: Visual Client Applications

Specifies whether spell-checking will be enabled for the control.

TEdit.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TEdit.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TEdit.Text Property

```
property Text: String
```

Available In: Visual Client Applications

Specifies the control's input value as a string.

TEdit.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TEdit.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TEdit.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TEdit.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TEdit.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TEdit.OnChange Event

```
property OnChange: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TEdit.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TEdit.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TEdit.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TEdit.OnEnter Event

```
property OnEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control obtains input focus.

TEdit.OnExit Event

```
property OnExit: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control loses input focus.

TEdit.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TEdit.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses a key or key combination.

TEdit.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TEdit.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user releases a key or key combination.

TEdit.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TEdit.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TEdit.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TEdit.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TEdit.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TEdit.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TEdit.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TEdit.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TEdit.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TEdit.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TEdit.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TEdit.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.69 TEditComboBox Component

Unit: WebEdits

Inherits From TEditComboControl

Available In: Visual Client Applications

The TEditComboBox component represents an edit combo box control. An edit combo box is a combo control that allows the user to directly enter an input value or select an input value from a drop-down list of values.

> **Note**
> If you do not want to allow for direct editing of the input value, please consider using a TButtonComboBox component instead.

Properties	Methods	Events
Alignment		OnButtonClick
AutoComplete		OnCaptureEnd
AutoDropDown		OnCaptureStart
AutoItemHeight		OnCapturing
Cursor		OnChange
DataColumn		OnClick
DataSet		OnContextMenu
Direction		OnDblClick
DropDownItemCount		OnDropDownHide
DropDownPosition		OnDropDownShow
DropDownVisible		OnEnter
Enabled		OnExit
Font		OnHide
Hint		OnKeyDown
ItemHeight		OnKeyPress
ItemIndex		OnKeyUp
Items		OnMouseDown
KeyPressInterval		OnMouseEnter
MaxLength		OnMouseLeave
ReadOnly		OnMouseMove
Sorted		OnMouseUp
SpellCheck		OnMove
TabOrder		OnShow

TabStop		OnSize
Text		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TEditComboBox.Alignment Property

```
property Alignment: TContentAlignment
```

Available In: Visual Client Applications

Specifies the alignment of the input value for the control.

TEditComboBox.AutoComplete Property

```
property AutoComplete: TAutoCompleteType
```

Available In: Visual Client Applications

Specifies how to handle auto-completion for the control. Auto-completion allows the browser to display a list of suggestions for input values, based upon earlier input values entered by the user.

TEditComboBox.AutoDropDown Property

```
property AutoDropDown: Boolean
```

Available In: Visual Client Applications

Specifies that the drop-down list of Items should automatically be shown when the user starts typing. The default value is False.

TEditComboBox.AutoItemHeight Property

```
property AutoItemHeight: Boolean
```

Available In: Visual Client Applications

Specifies that the displayed height of the drop-down items will automatically be set based upon the Font property settings. The default value is True.

TEditComboBox.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TEditComboBox.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TEditComboBox.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

TEditComboBox.Direction Property

```
property Direction: TContentDirection
```

Available In: Visual Client Applications

Specifies the direction in which the text is displayed/edited.

TEditComboBox.DropDownItemCount Property

```
property DropDownItemCount: Integer
```

Available In: Visual Client Applications

Specifies the number of visible items to display in the drop-down list of Items.

TEditComboBox.DropDownPosition Property

```
property DropDownPosition: TDropDownPosition
```

Available In: Visual Client Applications

Specifies where the drop-down list will be positioned.

TEditComboBox.DropDownVisible Property

```
property DropDownVisible: Boolean
```

Available In: Visual Client Applications

Indicates whether the drop-down list is visible.

TEditComboBox.Enabled Property

```
property Enabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TEditComboBox.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TEditComboBox.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TEditComboBox.ItemHeight Property

```
property ItemHeight: Integer
```

Available In: Visual Client Applications

Specifies the height, in pixels, of the items displayed in the drop-down list.

TEditComboBox.ItemIndex Property

```
property ItemIndex: Integer
```

Available In: Visual Client Applications

Specifies the index of the selected item in the drop-down list of Items, or -1 if there is no selected item.

TEditComboBox.Items Property

```
property Items: TStrings
```

Available In: Visual Client Applications

Specifies the items to use for the drop-down list.

TEditComboBox.KeyPressInterval Property

```
property KeyPressInterval: Integer
```

Available In: Visual Client Applications

Specifies the interval, in milliseconds, that is used by the control to combine user keystrokes into a search value that is then used for performing a near search on the Items property. Effectively, this means that the user has KeyPressInterval milliseconds in which to hit a key in order for the keystroke to be included as part of a near search. The default value is 300 milliseconds.

For example, if the user hits the "S", "M", and "I" keys within the KeyPressInterval property value, but hits the "T" key outside of the KeyPressInterval property, then the control will perform a near search using the value "SMI", followed by a near search using the value "T".

TEditComboBox.MaxLength Property

```
property MaxLength: Integer
```

Available In: Visual Client Applications

Specifies the maximum allowable length, in characters, of the Text property for the control. A value of 0 specifies an unlimited allowable length.

TEditComboBox.ReadOnly Property

```
property ReadOnly: Boolean
```

Available In: Visual Client Applications

Specifies whether the control's input value can be modified by the user. The default value is False.

> **Note**
> The input value can always be programmatically modified.

TEditComboBox.Sorted Property

```
property Sorted: Boolean
```

Available In: Visual Client Applications

Specifies whether the drop-down items will automatically be sorted. The default value is False.

TEditComboBox.SpellCheck Property

```
property SpellCheck: Boolean
```

Available In: Visual Client Applications

Specifies whether spell-checking will be enabled for the control.

TEditComboBox.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TEditComboBox.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TEditComboBox.Text Property

```
property Text: String
```

Available In: Visual Client Applications

Specifies the control's input value as a string.

TEditComboBox.OnButtonClick Event

```
property OnButtonClick: TClickEvent
```

Available In: Visual Client Applications

This event is triggered whenever the associated combo button is clicked.

Return True to allow the default click behavior and False to prevent the default click behavior from occurring.

TEditComboBox.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TEditComboBox.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TEditComboBox.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TEditComboBox.OnChange Event

```
property OnChange: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TEditComboBox.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TEditComboBox.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TEditComboBox.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TEditComboBox.OnDropDownHide Event

```
property OnDropDownHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the associated drop-down control is hidden.

TEditComboBox.OnDropDownShow Event

```
property OnDropDownShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the associated drop-down control is shown.

TEditComboBox.OnEnter Event

```
property OnEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control obtains input focus.

TEditComboBox.OnExit Event

```
property OnExit: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control loses input focus.

TEditComboBox.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TEditComboBox.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses a key or key combination.

TEditComboBox.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TEditComboBox.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user releases a key or key combination.

TEditComboBox.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TEditComboBox.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TEditComboBox.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TEditComboBox.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TEditComboBox.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TEditComboBox.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TEditComboBox.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TEditComboBox.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TEditComboBox.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TEditComboBox.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TEditComboBox.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TEditComboBox.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.70 TEditComboControl Component

Unit: WebEdits

Inherits From TDropDownEditControl

Available In: Visual Client Applications

The TEditComboControl control is the base class for edit combo controls, and contains all of the core combo functionality in the form of public methods and protected properties/events that descendant classes can use to create customized edit combo controls.

Properties	Methods	Events

10.71 TEditControl Component

Unit: WebEdits

Inherits From TInputControl

Available In: Visual Client Applications

The TEditControl control is the base class for text edit controls, and contains all of the core text edit functionality in the form of public methods and protected properties/events that descendant classes can use to create customized text edit controls.

Properties	Methods	Events
SelectionEnd	SelectAll	
SelectionStart	SelectNone	

TEditControl.SelectionEnd Property

```
property SelectionEnd: Integer
```

Available In: Visual Client Applications

Specifies the ending position (0-based) of the selected characters in the text. For example, if the control contains the text "Hello World", setting the SelectionStart property to 6 and the SelectionEnd property to 11 will result in the text "World" being selected.

TEditControl.SelectionStart Property

```
property SelectionStart: Integer
```

Available In: Visual Client Applications

Specifies the starting position (0-based) of the selected characters in the text. For example, if the control contains the text "Hello World", setting the SelectionStart property to 6 and the SelectionEnd property to 11 will result in the text "World" being selected.

TEditControl.SelectAll Method

```
procedure SelectAll
```

Available In: Visual Client Applications

Use this method to select all of the text content in the edit control.

TEditControl.SelectNone Method

```
procedure SelectNone
```

Available In: Visual Client Applications

Use this method to unselect any previously-selected text content in the edit control.

10.72 TElement Component

Unit: WebUI

Inherits From TPersistent

Available In: Visual Client Applications

The TElement class is the base element class for all UI elements, and contains all of the element functionality in the form of public methods and properties/events that control classes can use to create any type of control.

Properties	Methods	Events
AltKey	BeginRotation	
AlwaysOnTop	BeginUpdate	
Animations	BringToFront	
ApplyProperties	CancelRotation	
AutoSize	CanTab	
Background	Create	
Border	DefineLayout	
Button	EndUpdate	
ClientHeight	FindByName	
ClientWidth	FindByPos	
ComposedValue	FindFirstTab	
Constraints	FindLastTab	
Content	FindNextTab	
Controller	FindPriorTab	
ControllerElement	ForceUpdate	
Corners	HasTabs	
CtrlKey	Hide	
Cursor	IsControllerClass	
DefinedHeight	IsParentOf	
DefinedLeft	Minimize	
DefinedTop	RemoveFocus	
DefinedWidth	Restore	
Delegate	ScrollBy	
DisplayIndex	SendToBack	
DOMElement	SetFocus	
Elements	Show	
Enabled		
EventX		

EventY		
FocusEnabled		
Font		
FontSizeAnimation		
ForceDefaultCursor		
Format		
HasElements		
Height		
Hint		
HTMLContentEnabled		
ID		
InsetShadow		
InUpdate		
IsVisible		
KeyChar		
KeyCode		
Layout		
LayoutIndex		
Left		
Margins		
Minimized		
Name		
Opacity		
OutsetShadow		
OverflowX		
OverflowY		
Padding		
Parent		
PreserveFocus		
Scale		
ScrollEnabled		
ScrollHeight		
ScrollLeft		
ScrollTop		
ScrollWidth		
ShiftKey		
TabEnabled		

TabIndex		
TagName		
Top		
TotalHeight		
TotalWidth		
Visible		
WheelDelta		
Width		

TElement.AltKey Property

```
property AltKey: Boolean
```

Available In: Visual Client Applications

Indicates whether the Alt key was pressed during the last triggered event for this element.

TElement.AlwaysOnTop Property

```
property AlwaysOnTop: Boolean
```

Available In: Visual Client Applications

Specifies that the element should always be visually placed on top of any other element within the same container element.

TElement.Animations Property

```
property Animations: TAnimations
```

Available In: Visual Client Applications

Specifies the animations for the element.

TElement.ApplyProperties Property

```
property ApplyProperties: TElementProperties
```

Available In: Visual Client Applications

Specifies which properies of the element to apply to a UI element when applying the state of a control interface to a control.

TElement.AutoSize Property

```
property AutoSize: TAutoSize
```

Available In: Visual Client Applications

Specifies how (if at all) the element should automatically be sized based upon the Content and Format properties.

TElement.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the element.

TElement.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border of the element.

TElement.Button Property

```
property Button: Integer
```

Available In: Visual Client Applications

Indicates the code of the mouse button for the last triggered mouse event for this element.

TElement.ClientHeight Property

```
property ClientHeight: Integer
```

Available In: Visual Client Applications

Indicates the height of the client rectangle for the element.

TElement.ClientWidth Property

```
property ClientWidth: Integer
```

Available In: Visual Client Applications

Indicates the width of the client rectangle for the element.

TElement.ComposedValue Property

```
property ComposedValue: String
```

Available In: Visual Client Applications

Indicates the current composed value after the last triggered composition event for this element.

TElement.Constraints Property

```
property Constraints: TConstraints
```

Available In: Visual Client Applications

Indicates the dimensional constraints for the element.

TElement.Content Property

```
property Content: String
```

Available In: Visual Client Applications

Specifies the content of the element. The Font and Format properties control how the content is displayed.

TElement.Controller Property

```
property Controller: TInterfaceController
```

Available In: Visual Client Applications

Specifies the controller instance for the element. A controller instance is assigned to any element that serves as the base element for a TInterfaceController class descendant.

> **Note**
> The TInterfaceController class is never instantiated directly. It is used only as a bridge component between the TElement class and the TControl class.

TElement.ControllerElement Property

```
property ControllerElement: TElement
```

Available In: Visual Client Applications

Indicates the element, relative to the current element instance, that has its Controller property assigned. If the current element instance has its Controller property assigned, then this property will be equal to the current element instance.

TElement.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical corner radii for the element.

TElement.CtrlKey Property

```
property CtrlKey: Boolean
```

Available In: Visual Client Applications

Indicates whether the Control key was pressed during the last triggered event for this element.

TElement.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the type of cursor to show for the element.

Note
The ForceDefaultCursor property is useful for situations where the type of cursor is set to crAuto and the browser would normally show the cursor as a text caret for text selection, which is the case with any elements containing text as content.

TElement.DefinedHeight Property

```
property DefinedHeight: Integer
```

Available In: Visual Client Applications

Indicates the **defined** height for the element. The defined height is the last value that was directly assigned to the Height. Layout property changes may affect the value returned by the Height property, so this property is useful when you don't want the calculated height of an element.

TElement.DefinedLeft Property

```
property DefinedLeft: Integer
```

Available In: Visual Client Applications

Indicates the **defined** left position for the element. The defined left position is the last value that was directly assigned to the Left. Layout property changes may affect the value returned by the Left property, so this property is useful when you don't want the calculated left position of an element.

TElement.DefinedTop Property

```
property DefinedTop: Integer
```

Available In: Visual Client Applications

Indicates the **defined** top position for the element. The defined top position is the last value that was directly assigned to the Top. Layout property changes may affect the value returned by the Top property, so this property is useful when you don't want the calculated top position of an element.

TElement.DefinedWidth Property

```
property DefinedWidth: Integer
```

Available In: Visual Client Applications

Indicates the **defined** width for the element. The defined width is the last value that was directly assigned to the Width. Layout property changes may affect the value returned by the Height property, so this property is useful when you don't want the calculated width of an element.

TElement.Delegate Property

```
property Delegate: TElement
```

Available In: Visual Client Applications

Specifies that any focus/tabbing for the element should be delegated to the specified element. The specified element should be a child element of the current element instance. Delegation is useful for situations where a child element is the element that should be focused when the current element instance is focused.

TElement.DisplayIndex Property

```
property DisplayIndex: Integer
```

Available In: Visual Client Applications

Specifies the display index, or visual stacking index, of the element within its parent container element.

TElement.DOMElement Property

```
property DOMElement: THTMLElement
```

Available In: Visual Client Applications

Contains a reference to the underlying DOM element associated with the element.

TElement.Elements Property

```
property Elements: TElements
```

Available In: Visual Client Applications

Contains references to all child elements that have the current element instance assigned as its Parent property. Element parentage automatically implies element ownership, so any child elements of the current element instance will automatically be disposed of when the current element instance is destroyed.

Warning
This property will be nil if no child elements have been assigned to the element.

TElement.Enabled Property

```
property Enabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the element is enabled or disabled.

TElement.EventX Property

```
property EventX: Integer
```

Available In: Visual Client Applications

Indicates the horizontal position of the mouse button/touch for the last triggered mouse/touch event for this element.

TElement.EventY Property

```
property EventY: Integer
```

Available In: Visual Client Applications

Indicates the vertical position of the mouse button/touch for the last triggered mouse/touch event for this element.

TElement.FocusEnabled Property

```
property FocusEnabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the element should be focusable.

TElement.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies font to use for the element content.

TElement.FontSizeAnimation Property

```
property FontSizeAnimation: TAnimation
```

Available In: Visual Client Applications

Specifies the animation properties for the element font's Size property.

TElement.ForceDefaultCursor Property

```
property ForceDefaultCursor: Boolean
```

Available In: Visual Client Applications

Specifies that the cursor should be dynamically changed to crDefault if the Cursor property is set to crAuto and the browser would normally show the cursor as a text caret for text selection, which is the case with any elements containing text as content.

TElement.Format Property

```
property Format: TFormat
```

Available In: Visual Client Applications

Specifies the content formatting to use for the element's content.

TElement.HasElements Property

```
property HasElements: Boolean
```

Available In: Visual Client Applications

Indicates whether the Elements is nil, and if not, whether the there are any child elements in the list of elements.

TElement.Height Property

```
property Height: Integer
```

Available In: Visual Client Applications

Indicates the current height of the element, and can be used to specify the **defined** height for the element. The defined height is the last value that was directly assigned to the Height property. Layout property changes may affect the value returned by the Height property.

TElement.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies a popup hint to display during a mouse-over in a desktop browser or a touch in a mobile browser.

TElement.HTMLContentEnabled Property

```
property HTMLContentEnabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the element can contain HTML content. If this property is False, then any text assigned to the Content property will treated as plain text. If this property is True, then any text assigned to the Content property will be treated as HTML.

Note
Enabling this property modifies how the UI layout functionality treats the content with respect to measurement and display.

TElement.ID Property

```
property ID: String
```

Available In: Visual Client Applications

Specifies a unique DOM ID for the element.

TElement.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Available In: Visual Client Applications

Specifies the inset shadow for the element. The inset shadow appears within the bounds of the client rectangle for the element.

TElement.InUpdate Property

```
property InUpdate: Boolean
```

Available In: Visual Client Applications

Indicates whether the element, or any of its parent elements, is currently in a batch update. An element is in a batch update if the BeginUpdate method is called on the element or any of its parent elements. When an element is in a batch update, it doesn't apply any changes to any of its properties until the EndUpdate method is called on the element or any of its parent elements, and the InUpdate property returns False.

Note
Updates are reference-counted so calls to the BeginUpdate method increment the reference count, and calls to the EndUpdate method decrement the reference count.

TElement.IsVisible Property

```
property IsVisible: Boolean
```

Available In: Visual Client Applications

Indicates whether the element, and **all** of its parent elements, are visible. This is in contrast to the Visible property, which only indicates whether the current element is visible.

TElement.KeyChar Property

```
property KeyChar: Char
```

Available In: Visual Client Applications

Indicates the character that represents the keystroke for the last triggered keypress event for this element.

TElement.KeyCode Property

```
property KeyCode: Integer
```

Available In: Visual Client Applications

Indicates the code of the keystroke for the last triggered key event for this element.

TElement.Layout Property

```
property Layout: TLayout
```

Available In: Visual Client Applications

Specifies the layout for the element.

TElement.LayoutIndex Property

```
property LayoutIndex: Integer
```

Available In: Visual Client Applications

Specifies the layout index of the element. The layout index determines the order in which the child elements of a container element are positioned and sized using the layout management functionality for UI elements.

TElement.Left Property

```
property Left: Integer
```

Available In: Visual Client Applications

Indicates the current left position of the element, and can be used to specify the **defined** left position for the element. The defined left position is the last value that was directly assigned to the Left property. Layout property changes may affect the value returned by the Left property.

TElement.Margins Property

```
property Margins: TMargins
```

Available In: Visual Client Applications

Specifies the margins to be used for the element when the element is being positioned/sized using the layout management functionality.

TElement.Minimized Property

```
property Minimized: Boolean
```

Available In: Visual Client Applications

Indicates whether the element is currently minimized.

TElement.Name Property

```
property Name: String
```

Available In: Visual Client Applications

Specifies the name of the element. Element names must be unique within a an interface state and within the base and child elements of a visual control.

TElement.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the element, from 0 (transparent) to 100 (opaque).

TElement.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Available In: Visual Client Applications

Specifies the outset shadow for the element. The outset shadow appears behind the bounds of the element.

TElement.OverflowX Property

```
property OverflowX: TOverflowType
```

Available In: Visual Client Applications

Specifies whether or not to show a native horizontal browser scrollbar for the element if the width of its contents and/or child elements exceeds the width of the client rectangle for the element.

Note
This property should be left at its default value of otHidden for most elements. This element property is only used for specifying the scrollbars of the application viewport.

TElement.OverflowY Property

```
property OverflowY: TOverflowType
```

Available In: Visual Client Applications

Specifies whether or not to show a native vertical browser scrollbar for the element if the height of its contents and/or child elements exceeds the height of the client rectangle for the element.

> **Note**
> This property should be left at its default value of otHidden for most elements. This element property is only used for specifying the scrollbars of the application viewport.

TElement.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies any padding for the element. The padding of an element shrinks the width and height of the client rectangle for the element.

TElement.Parent Property

```
property Parent: TElement
```

Available In: Visual Client Applications

Specifies the parent element for the element. The only element whose Parent property is always nil is the interface manager's root element.

TElement.PreserveFocus Property

```
property PreserveFocus: Boolean
```

Available In: Visual Client Applications

Specifies whether the element should preserve focus on an existing focused element if an attempt is made to focus the element and the FocusEnabled property is False. This property is used by non-focusable elements such as scrollbar control elements to make sure that clicking on the elements does not cause any currently-focused elements to lose focus.

TElement.Scale Property

```
property Scale: Integer
```

Available In: Visual Client Applications

Specifies the scale of the element, from 1 to 1000. The scale affects how the element is drawn, but does not affect the actual layout of the element.

TElement.ScrollEnabled Property

```
property ScrollEnabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the element wants mouse wheel and touch scroll events routed to it. Whenever a mouse wheel or touch scroll event occurs, the event manager will automatically route such events to the first container element that has its ScrollEnabled property set to True.

TElement.ScrollHeight Property

```
property ScrollHeight: Integer
```

Available In: Visual Client Applications

Indicates the total height of the element's content and/or its child elements. If an element's content height is greater than its Height property, then you can use the ScrollTop property to programmatically scroll the element vertically, or to find out the current vertical scroll position.

TElement.ScrollLeft Property

```
property ScrollLeft: Integer
```

Available In: Visual Client Applications

Specifies the horizontal scroll position for the element. If an element's ScrollWidth property is greater than its Width property, then you can use this property to programmatically scroll the element horizontally, or to find out the current horizontal scroll position.

TElement.ScrollTop Property

```
property ScrollTop: Integer
```

Available In: Visual Client Applications

Specifies the vertical scroll position for the element. If an element's ScrollHeight property is greater than its Height property, then you can use this property to programmatically scroll the element vertically, or to determine the current vertical scroll position.

TElement.ScrollWidth Property

```
property ScrollWidth: Integer
```

Available In: Visual Client Applications

Indicates the total width of the element's content and/or its child elements. If an element's content width is greater than its Width property, then you can use the ScrollLeft property to programmatically scroll the element horizontally, or to determine the current horizontal scroll position.

TElement.ShiftKey Property

```
property ShiftKey: Boolean
```

Available In: Visual Client Applications

Indicates whether the Shift key was pressed during the last triggered event for this element.

TElement.TabEnabled Property

```
property TabEnabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the element will take part in any tabbing order within its parent container element.

TElement.TabIndex Property

```
property TabIndex: Integer
```

Available In: Visual Client Applications

Specifies the tab index of the element within its parent container element.

TElement.TagName Property

```
property TagName: String
```

Available In: Visual Client Applications

Indicates the HTML tag name associated with the element at runtime. HTML tag names are used at runtime to create elements using a mapping between the tag name and a given TElement class type.

TElement.Top Property

```
property Top: Integer
```

Available In: Visual Client Applications

Indicates the current top position of the element, and can be used to specify the **defined** top position for the element. The defined top position is the last value that was directly assigned to the Top property. Layout property changes may affect the value returned by the Top property.

TElement.TotalHeight Property

```
property TotalHeight: Integer
```

Available In: Visual Client Applications

Indicates the total height of the element, including its top and bottom margins.

TElement.TotalWidth Property

```
property TotalWidth: Integer
```

Available In: Visual Client Applications

Indicates the total width of the element, including its left and right margins.

TElement.Visible Property

```
property Visible: Boolean
```

Available In: Visual Client Applications

Specifies whether the element is visible or not.

TElement.WheelDelta Property

```
property WheelDelta: Integer
```

Available In: Visual Client Applications

Indicates the mouse wheel delta for the last triggered mouse wheel event for this element.

TElement.Width Property

```
property Width: Integer
```

Available In: Visual Client Applications

Indicates the current width of the element, and can be used to specify the **defined** width for the element. The defined width is the last value that was directly assigned to the Width property. Layout property changes may affect the value returned by the Width property.

TElement.BeginRotation Method

```
procedure BeginRotation(AStyle: TAnimationStyle; AInterval:
      Integer)
```

Available In: Visual Client Applications

Use this method to begin rotating the contents of the element using the animation properties specified by the AStyle and AInterval parameters. The rotation animation will continue until the CancelRotation method is called.

> **Note**
> The component library uses this functionality for animating the rotation of font icons in progress dialogs.

TElement.BeginUpdate Method

```
procedure BeginUpdate
```

Available In: Visual Client Applications

Use this method to begin a batch update on an element. An element is in a batch update if the BeginUpdate method is called on the element, or any of its parent elements. When an element is in a batch update, it doesn't apply any changes to any of its properties until the EndUpdate method is called on the element or any of its parent elements, and the InUpdate property returns False.

Note
Updates are reference-counted so calls to the BeginUpdate method increment the reference count, and calls to the EndUpdate method decrement the reference count.

TElement.BringToFront Method

```
procedure BringToFront
```

Available In: Visual Client Applications

Use this method to bring the element to the front of the visual stacking order. An element in the front will have a DisplayIndex property equal to one less than the number of child elements in the parent container element.

TElement.CancelRotation Method

```
procedure CancelRotation
```

Available In: Visual Client Applications

Use this method to cancel the rotation animation started using the BeginRotation method.

Note
The component library uses this functionality for animating the rotation of font icons in progress dialogs.

TElement.CanTab Method

```
function CanTab(AClass: TInterfaceControllerClass=nil): Boolean
```

Available In: Visual Client Applications

Use this method to determine if an element, with or without a particular associated Controller instance, is visible, enabled, and can be focused, or contains a child element that can be focused.

TElement.Create Method

```
constructor Create(const AName: String; AParent: TElement=nil;
      const ATagName: String='')
```

Available In: Visual Client Applications

Use this method to create a new instance of the TElement class. The AName parameter indicates the name of the element, the optional AParent parameter indicates the parent of the element, and the optional ATagName parameter indicates the HTML tag name to associate with the element. The HTML tag name is used to create the underlying browser DOM element that will be managed by the element at runtime.

TElement.DefineLayout Method

```
procedure DefineLayout
```

Available In: Visual Client Applications

Use this method to assign the current bounds of the element to the Left, Top, Width, and Height properties of the element.

This method is useful in situations where an element has been positioned or sized according to its Layout properties, but you wish to have the bounds persist even after modifying the layout properties so that they no longer position or size the element in the same way.

TElement.EndUpdate Method

```
procedure EndUpdate
```

Available In: Visual Client Applications

Use this method to end a batch update on an element. An element is in a batch update if the BeginUpdate method is called on the element or any of its parent elements. When an element is in a batch update, it doesn't apply any changes to any of its properties until the EndUpdate method is called on the element or any of its parent elements, and the InUpdate property returns False.

Note
Updates are reference-counted so calls to the BeginUpdate method increment the reference count, and calls to the EndUpdate method decrement the reference count.

TElement.FindByName Method

```
function FindByName(const AName: String): TElement
```

Available In: Visual Client Applications

Use this method to find an element with a specified name. The search for the element includes the current element instance.

> **Note**
> This is a recursive method, and will search through grandchildren, great-grandchildren, etc. for an element with the specified name.

TElement.FindByPos Method

```
function FindByPos(X,Y: Integer): TElement
```

Available In: Visual Client Applications

Use this method to find an element, if one exists, at the specified coordinates. The search for the element includes the current element instance, and is performed using the visual stacking order defined by the DisplayIndex property of each element. Elements with a smaller display index (in front of other elements) will be found instead of elements with a larger display index, even if both elements have the same bounds.

Note
This is a recursive method, and will search through grandchildren, great-grandchildren, etc. for an element with the specified coordinates.

TElement.FindFirstTab Method

```
function FindFirstTab(AClass: TInterfaceControllerClass=nil):
      TElement
```

Available In: Visual Client Applications

Use this method to find the first child element in TabIndex order, with or without a particular associated Controller instance, whose CanTab method returns True.

TElement.FindLastTab Method

```
function FindLastTab(AClass: TInterfaceControllerClass=nil):
      TElement
```

Available In: Visual Client Applications

Use this method to find the last child element in TabIndex order, with or without a particular associated Controller instance, whose CanTab method returns True.

TElement.FindNextTab Method

```
function FindNextTab(AElement: TElement; AWrap: Boolean=False;
      AClass: TInterfaceControllerClass=nil): TElement
```

Available In: Visual Client Applications

Use this method to find the next child element in TabIndex order, with or without a particular associated Controller instance, whose CanTab method returns True.

TElement.FindPriorTab Method

```
function FindPriorTab(AElement: TElement; AWrap: Boolean=False;
      AClass: TInterfaceControllerClass=nil): TElement
```

Available In: Visual Client Applications

Use this method to find the prior child element in TabIndex order, with or without a particular associated Controller instance, whose CanTab method returns True.

TElement.ForceUpdate Method

```
procedure ForceUpdate
```

Available In: Visual Client Applications

Use this method to force the element to perform a layout update when in the middle of a BeginUpdate/EndUpdate block. Normally, an element will not update its layout bounds when an update block is in effect. A forced layout update will ensure that the element's layout bounds are updated to reflect any layout changes applied to the element within the update block.

TElement.HasTabs Method

```
function HasTabs(AClass: TInterfaceControllerClass=nil): Boolean
```

Available In: Visual Client Applications

Use this method to determine if an element, with or without a particular associated Controller instance, contains a child element that can be focused.

TElement.Hide Method

```
procedure Hide
```

Available In: Visual Client Applications

Use this method to set the Visible property to False and hide the element.

TElement.IsControllerClass Method

```
function IsControllerClass(AClass: TInterfaceControllerClass;
      ANameRequired: Boolean=False): Boolean
```

Available In: Visual Client Applications

Use this method to determine if the element's associated Controller is the specified controller class type.

TElement.IsParentOf Method

```
function IsParentOf(AElement: TElement): Boolean
```

Available In: Visual Client Applications

Use this method to determine if the element is the parent of the specified element.

> **Note**
> This is a recursive method, and will search **all** child elements, even grandchildren, great-grandchildren, etc. to determine if any of the child elements are the parent of the specified element.

TElement.Minimize Method

```
function Minimize: Boolean
```

Available In: Visual Client Applications

Use this method to minimize the element. Minimizing an element saves the current width and height of the element so that it can be restored later using the Restore method, and sets the Minimized to True.

This method will return True if the element was not already minimized and False if it was already minimized.

TElement.RemoveFocus Method

```
procedure RemoveFocus
```

Available In: Visual Client Applications

Use this method to remove focus from the element. If the element is not focused, then this method does nothing.

TElement.Restore Method

```
function Restore: Boolean
```

Available In: Visual Client Applications

Use this method to restore a minimized element. A minimized element is an element whose Minimized property is True.

This method will return True if the element was minimized and False if it was not minimized.

TElement.ScrollBy Method

```
procedure ScrollBy(X,Y: Integer)
```

Available In: Visual Client Applications

If an element's content width and/or height is greater than its Width and Height properties, then you can use this method to scroll the contents of the element horizontally, vertically, or both. The X and Y values represent the number of pixels to scroll the contents by, and may be negative values for scrolling backward.

TElement.SendToBack Method

```
procedure SendToBack
```

Available In: Visual Client Applications

Use this method to send the element to the back of the visual stacking order. An element in the back will have a DisplayIndex property equal to 0.

TElement.SetFocus Method

```
procedure SetFocus
```

Available In: Visual Client Applications

Use this method to set focus to the element. If the element's FocusEnabled property is set to True, then the element will be focused. If the element's PreserveFocus property is set to False, then focus will be removed from whatever element is currently focused.

TElement.Show Method

```
procedure Show
```

Available In: Visual Client Applications

Use this method to set the Visible to True and show the element.

10.73 TElementAttribute Component

Unit: WebUI

Inherits From TPersistent

Available In: Visual Client Applications

The TElementAttribute class represents an attribute for a UI element or control. It is the base class for all attribute classes like the TBackground, TFont, and TLayout classes and contains the base functionality for loading and change management for element attributes.

Properties	Methods	Events
	Create	
	GetStyle	

TElementAttribute.Create Method

```
constructor Create(AElement: TElement; AParent:
      TElementAttribute)
```

Available In: Visual Client Applications

Use this method to create a new instance of the TElementAttribute class. The AElement parameter indicates the element instance that will manage the attribute, and the AParent parameter indicates the parent attribute, if any, that the attribute is contained within. The parent attribute is used to aggregate change management at the outermost attribute so as to avoid excessively triggering change notifications in the element.

TElementAttribute.GetStyle Method

```
function GetStyle: String
```

Available In: Visual Client Applications

Use this method at run-time to get a string containing the CSS style data for the element attribute.

> **Note**
> This method is not available at design-time for element attributes.

10.74 TElementProperties Component

Unit: WebUI

Inherits From TPersistent

Available In: Visual Client Applications

The TElementProperties class specifies which properties of an element are applied when a control interface state is applied to the elements of a control instance.

Properties	Methods	Events
AlwaysOnTop		
AutoSize		
Background		
Border		
Constraints		
Content		
Corners		
Cursor		
DisplayIndex		
Font		
FontColor		
FontSize		
FontStyle		
Format		
Height		
InsetShadow		
Layout		
LayoutIndex		
Left		
Margins		
Opacity		
OutsetShadow		
Padding		
Top		
Visible		
Width		

TElementProperties.AlwaysOnTop Property

```
property AlwaysOnTop: Boolean
```

Available In: Visual Client Applications

Specifies that the AlwaysOnTop property should be applied.

TElementProperties.AutoSize Property

```
property AutoSize: Boolean
```

Available In: Visual Client Applications

Specifies that the AutoSize property should be applied.

TElementProperties.Background Property

```
property Background: Boolean
```

Available In: Visual Client Applications

Specifies that the Background property should be applied.

TElementProperties.Border Property

```
property Border: Boolean
```

Available In: Visual Client Applications

Specifies that the Border property should be applied.

TElementProperties.Constraints Property

```
property Constraints: Boolean
```

Available In: Visual Client Applications

Specifies that the Constraints property should be applied.

TElementProperties.Content Property

```
property Content: Boolean
```

Available In: Visual Client Applications

Specifies that the Content property should be applied.

TElementProperties.Corners Property

```
property Corners: Boolean
```

Available In: Visual Client Applications

Specifies that the Corners property should be applied.

TElementProperties.Cursor Property

```
property Cursor: Boolean
```

Available In: Visual Client Applications

Specifies that the Cursor property should be applied.

TElementProperties.DisplayIndex Property

```
property DisplayIndex: Boolean
```

Available In: Visual Client Applications

Specifies that the DisplayIndex property should be applied.

TElementProperties.Font Property

```
property Font: Boolean
```

Available In: Visual Client Applications

Specifies that the Font property should be applied.

TElementProperties.FontColor Property

```
property FontColor: Boolean
```

Available In: Visual Client Applications

Specifies that the Font.Color property should be applied.

TElementProperties.FontSize Property

```
property FontSize: Boolean
```

Available In: Visual Client Applications

Specifies that the Font.Size property should be applied.

TElementProperties.FontStyle Property

```
property FontStyle: Boolean
```

Available In: Visual Client Applications

Specifies that the Font.Style property should be applied.

TElementProperties.Format Property

```
property Format: Boolean
```

Available In: Visual Client Applications

Specifies that the Format property should be applied.

TElementProperties.Height Property

```
property Height: Boolean
```

Available In: Visual Client Applications

Specifies that the Height property should be applied.

TElementProperties.InsetShadow Property

```
property InsetShadow: Boolean
```

Available In: Visual Client Applications

Specifies that the InsetShadow property should be applied.

TElementProperties.Layout Property

```
property Layout: Boolean
```

Available In: Visual Client Applications

Specifies that the Layout property should be applied.

TElementProperties.LayoutIndex Property

```
property LayoutIndex: Boolean
```

Available In: Visual Client Applications

Specifies that the LayoutIndex property should be applied.

TElementProperties.Left Property

```
property Left: Boolean
```

Available In: Visual Client Applications

Specifies that the Left property should be applied.

TElementProperties.Margins Property

```
property Margins: Boolean
```

Available In: Visual Client Applications

Specifies that the Margins property should be applied.

TElementProperties.Opacity Property

```
property Opacity: Boolean
```

Available In: Visual Client Applications

Specifies that the Opacity property should be applied.

TElementProperties.OutsetShadow Property

```
property OutsetShadow: Boolean
```

Available In: Visual Client Applications

Specifies that the OutsetShadow property should be applied.

TElementProperties.Padding Property

```
property Padding: Boolean
```

Available In: Visual Client Applications

Specifies that the Padding property should be applied.

TElementProperties.Top Property

```
property Top: Boolean
```

Available In: Visual Client Applications

Specifies that the Top property should be applied.

TElementProperties.Visible Property

```
property Visible: Boolean
```

Available In: Visual Client Applications

Specifies that the Visible property should be applied.

TElementProperties.Width Property

```
property Width: Boolean
```

Available In: Visual Client Applications

Specifies that the Width property should be applied.

10.75 TElements Component

Unit: WebUI

Inherits From TPersistent

Available In: Visual Client Applications

The TElements class is a container for the child elements of an element.

Properties	Methods	Events
Count	Create	
DisplayElement		
Element		
TabCount		
TabElement		

TElements.Count Property

```
property Count: Integer
```

Available In: Visual Client Applications

Indicates the number of child elements.

TElements.DisplayElement Property

```
property DisplayElement[Index: Integer]: TElement
```

Available In: Visual Client Applications

Accesses child elements by their DisplayIndex property.

TElements.Element Property

```
property Element[Index: Integer]: TElement
```

Available In: Visual Client Applications

Accesses child elements by their LayoutIndex property.

TElements.TabCount Property

```
property TabCount: Integer
```

Available In: Visual Client Applications

Indicates the number of focusable and tabable child elements.

TElements.TabElement Property

```
property TabElement[Index: Integer]: TElement
```

Available In: Visual Client Applications

Accesses child elements by their TabIndex property.

TElements.Create Method

```
constructor Create(AElement: TElement)
```

Available In: Visual Client Applications

Use this method to create a new instance of the TElements class. The AElement parameter indicates the parent element instance that will manage the elements.

10.76 TFileComboBox Component

Unit: WebEdits

Inherits From TEditComboControl

Available In: Visual Client Applications

The TFileComboBox component represents a file combo box control. A file combo box is a combo box control that allows the user to select a file from the browser's file selection dialog.

Properties	Methods	Events
AcceptTypes		OnAnimationComplete
Cursor		OnAnimationsComplete
Enabled		OnButtonClick
FilePath		OnCaptureEnd
Font		OnCaptureStart
Hint		OnCapturing
Icon		OnChange
ReadOnly		OnClick
TabOrder		OnContextMenu
TabStop		OnDblClick
Text		OnEnter
		OnExit
		OnHide
		OnKeyDown
		OnKeyPress
		OnKeyUp
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TFileComboBox.AcceptTypes Property

```
property AcceptTypes: TStrings
```

Available In: Visual Client Applications

Specifies a list of MIME types or file extensions for filtering the list of files that are shown to the user when the file combo box control's drop-down button is clicked.

TFileComboBox.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TFileComboBox.Enabled Property

```
property Enabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TFileComboBox.FilePath Property

```
property FilePath: String
```

Available In: Visual Client Applications

If a file has been selected by the user, this property will indicate the path of the selected file.

> **Note**
> Some browsers/operating systems may not return a path for the file selected by the user in order to prevent local machine information from being leaked to the web server. In such a case, this property will be blank ('').

TFileComboBox.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TFileComboBox.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TFileComboBox.Icon Property

```
property Icon: TIconProperties
```

Available In: Visual Client Applications

Specifies the properties of the icon used with the button in the control.

TFileComboBox.ReadOnly Property

```
property ReadOnly: Boolean
```

Available In: Visual Client Applications

Specifies whether the control's input value can be modified by the user. The default value is False.

Note
The input value can always be programmatically modified.

TFileComboBox.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TFileComboBox.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TFileComboBox.Text Property

```
property Text: String
```

Available In: Visual Client Applications

Specifies the control's input value as a string.

TFileComboBox.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TFileComboBox.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TFileComboBox.OnButtonClick Event

```
property OnButtonClick: TClickEvent
```

Available In: Visual Client Applications

This event is triggered whenever the associated combo button is clicked.

Return True to allow the default click behavior and False to prevent the default click behavior from occurring.

TFileComboBox.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TFileComboBox.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TFileComboBox.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TFileComboBox.OnChange Event

```
property OnChange: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TFileComboBox.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TFileComboBox.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TFileComboBox.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TFileComboBox.OnEnter Event

```
property OnEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control obtains input focus.

TFileComboBox.OnExit Event

```
property OnExit: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control loses input focus.

TFileComboBox.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TFileComboBox.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses a key or key combination.

TFileComboBox.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TFileComboBox.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user releases a key or key combination.

TFileComboBox.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TFileComboBox.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TFileComboBox.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TFileComboBox.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TFileComboBox.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TFileComboBox.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TFileComboBox.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TFileComboBox.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TFileComboBox.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TFileComboBox.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TFileComboBox.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TFileComboBox.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.77 TFileInputElement Component

Unit: WebUI

Inherits From TInputElement

Available In: Visual Client Applications

The TFileInputElement class is the element class for file input elements, and contains all of the file input functionality in the form of public methods and properties/events that control classes can use to create file upload controls.

> **Note**
> This element does not provide support for file uploads at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events
AcceptTypes	Click	

TFileInputElement.AcceptTypes Property

```
property AcceptTypes: String
```

Available In: Visual Client Applications

Specifies a comma-delimited list of MIME types or file extensions for filtering the list of files that are shown to the user when the file input element is clicked.

TFileInputElement.Click Method

```
procedure Click
```

Available In: Visual Client Applications

Use this method to programmatically simulate a click on the element.

> **Note**
> Due to security restrictions, an application can only successfully call this method if the calling code was initiated by a user interfaction, such as a mouse click or touch.

10.78 TFileStream Component

Unit: WebSrvr

Inherits From TStream

Available In: Server Applications

The TFileStream class is used to create and/or open files and perform read/write operations using the ancestor TStream class properties/methods.

Properties	Methods	Events
FileName	Create	
LastWriteTime	Flush	
	Lock	
	Unlock	

TFileStream.FileName Property

```
property FileName: String
```

Available In: Server Applications

Indicates the complete file name of the file being accessed by the file stream.

TFileStream.LastWriteTime Property

```
property LastWriteTime: DateTime
```

Available In: Server Applications

Indicates the last time that the file being accessed by the file stream was written to.

TFileStream.Create Method

```
constructor Create(const AFileName: String; const Mode: Integer;
      const Rights: Integer=0)
```

Available In: Server Applications

Use this method to create a new instance of the TFileStream class for accessing the specified file.

> **Note**
> The specified file name should contain an absolute path. Relative paths are permitted, but they rely on the current working directory. The current working directory is a per-process setting and is not reliable in a multi-threaded setting like the web server.

The mode parameter is specified by OR'ing a file create/open mode with a file sharing mode.

The valid create/open modes are:

Open Mode	Description
FILE_CREATE ($FF00)	Specifies that the file should be created, regardless of whether the file already exists.
FILE_CREATE_EXCLUSIVE ($0004)	OR'd with FILE_CREATE, specifies that the file should only be created if it doesn't already exist. If the file already exists, then an exception is raised.
FILE_OPEN_READ ($0000)	Specifies that the file should be opened with read capabilities.
FILE_OPEN_WRITE ($0001)	Specifies that the file should be opened with write capabilities.
FILE_OPEN_READWRITE ($0002)	Specifies that the file should be opened with read and write capabilities.

The valid sharing modes are:

Sharing Mode	Description
FILE_SHARE_EXCLUSIVE ($0010)	Specifies that no other processes will be able to open the file until the file stream is destroyed.
FILE_SHARE_DENYWRITE ($0020)	Specifies that no other processes will be able to write to the file until the file stream is destroyed.
FILE_SHARE_DENYNONE ($0040)	Specifies that other processes will be able to open, read, and write to the file.

> **Note**
> The rights parameter is not used under Windows.

TFileStream.Flush Method

```
procedure Flush
```

Available In: Server Applications

Use this method to request that the operating system flush any file writes that have been buffered for the file by the file system but not physically written to disk yet. The file system in most operating systems is buffered so that the most recently accessed areas of a file are cached in memory instead of being written to disk. This also applies to file writes, which will often be kept in memory until all open instances of the file are closed or the memory needs to be used for other purposes.

TFileStream.Lock Method

```
function Lock(const Offset: Integer; const Count: Integer):
      Boolean
```

Available In: Server Applications

Use this method to lock a region of the file being accessed by the file stream. The method will return True if the lock succeeds, and False if the lock fails.

Note
You are allowed to lock a region of the file that starts, or extends, beyond the physical end of the file.

TFileStream.Unlock Method

```
function Unlock(const Offset: Integer; const Count: Integer):
      Boolean
```

Available In: Server Applications

Use this method to unlock a region of the file being accessed by the file stream. The region must have been previously locked using the TFileStream Lock method. The method will return True if the unlock succeeds, and False if the unlock fails.

10.79 TFill Component

Unit: WebUI

Inherits From TElementAttribute

Available In: Visual Client Applications

The TFill class represents the background fill of a UI element or control. Background fills can be solid colors (including transparent) or gradients.

Properties	Methods	Events
Color	SetToDefault	
FillType		
Gradient		

TFill.Color Property

```
property Color: TColor
```

Available In: Visual Client Applications

Specifies the color of the background fill.

TFill.FillType Property

```
property FillType: TFillType
```

Available In: Visual Client Applications

Specifies the type of background fill.

TFill.Gradient Property

```
property Gradient: TGradient
```

Available In: Visual Client Applications

Specifies a gradient background fill.

TFill.SetToDefault Method

```
procedure SetToDefault
```

Available In: Visual Client Applications

Use this method to reset the background fill's properties to their default values.

10.80 TFont Component

Unit: WebUI

Inherits From TElementAttribute

Available In: Visual Client Applications

The TFont class represents the font to use for the content of a UI element or control.

Properties	Methods	Events
Color	SetToDefault	
GenericFamily		
LineHeight		
Name		
Size		
Style		

TFont.Color Property

```
property Color: TColor
```

Available In: Visual Client Applications

Specifies the color of the font. The default value is clBlack.

TFont.GenericFamily Property

```
property GenericFamily: TGenericFontFamily
```

Available In: Visual Client Applications

Specifies the generic font family for the font.

> **Note**
> This property is automatically populated at design-time when the Name property is changed.

TFont.LineHeight Property

```
property LineHeight: Integer
```

Available In: Visual Client Applications

Indicates the calculated line height, in pixels, based upon the Size property.

TFont.Name Property

```
property Name: String
```

Available In: Visual Client Applications

Specifies the name of the font. The default value is "Arial".

TFont.Size Property

```
property Size: Integer
```

Available In: Visual Client Applications

Specifies the size, in pixels, of the font. The default value is 16.

TFont.Style Property

```
property Style: TFontStyle
```

Available In: Visual Client Applications

Specifies the style of the font.

TFont.SetToDefault Method

```
procedure SetToDefault
```

Available In: Visual Client Applications

Use this method to reset the font's properties to their default values.

10.81 TFontIcon Component

Unit: WebCtrls

Inherits From TComponent

Available In: Visual Client Applications

The TFontIcon component represents the font icon for a TIcon control. It includes properties that control the attributes of the font icon, such as the size and color of the icon.

Properties	Methods	Events
AutoSize		
Color		
Size		

TFontIcon.AutoSize Property

```
property AutoSize: Boolean
```

Available In: Visual Client Applications

Specifies that the font icon should automatically be sized based upon the TIconProperties Height property.

TFontIcon.Color Property

```
property Color: TColor
```

Available In: Visual Client Applications

Specifies the color of the font icon. This property defaults to the pre-defined font icon color of the icon specified by the IconName property, and may change when the IconName property is changed.

TFontIcon.Size Property

```
property Size: Integer
```

Available In: Visual Client Applications

Specifies the size, in pixels, of the font icon. This property defaults to the pre-defined font icon color of the icon specified by the IconName property, and may change when the IconName property is changed.

10.82 TFontStyle Component

Unit: WebUI

Inherits From TElementAttribute

Available In: Visual Client Applications

The TFontStyle class represents the style attributes of the font to use for the content of a UI element or control.

Properties	Methods	Events
Bold	SetToDefault	
Italic		
Strikeout		
Underline		

TFontStyle.Bold Property

```
property Bold: Boolean
```

Available In: Visual Client Applications

Specifies that the font should be bold. The default value is False.

TFontStyle.Italic Property

```
property Italic: Boolean
```

Available In: Visual Client Applications

Specifies that the font should be italicized. The default value is False.

TFontStyle.Strikeout Property

```
property Strikeout: Boolean
```

Available In: Visual Client Applications

Specifies that the font should have a line drawn through it. The default value is False.

TFontStyle.Underline Property

```
property Underline: Boolean
```

Available In: Visual Client Applications

Specifies that the font should be underlined. The default value is False.

TFontStyle.SetToDefault Method

```
procedure SetToDefault
```

Available In: Visual Client Applications

Use this method to reset the font style's properties to their default values.

10.83 TForm Component

Unit: WebForms

Inherits From TFormControl

Available In: Visual Client Applications

The TForm component represents a basic form control. Please see the Creating and Showing Forms for more information on using forms.

Properties	Methods	Events
ActivateOnClick		OnAnimationComplete
Background		OnAnimationsComplete
Border		OnCaptureEnd
Corners		OnCaptureStart
Cursor		OnCapturing
InsetShadow		OnClick
Opacity		OnClientSize
OutsetShadow		OnClose
Padding		OnCloseQuery
ScrollBars		OnContextMenu
ScrollSupport		OnDblClick
		OnHide
		OnKeyDown
		OnKeyPress
		OnKeyUp
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMouseWheel
		OnMove
		OnScroll
		OnScrollSize
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd

		OnTouchMove
		OnTouchScroll
		OnTouchStart

TForm.ActivateOnClick Property

```
property ActivateOnClick: Boolean
```

Available In: Visual Client Applications

Specifies whether the form should automatically be brought to the front when it, or any child controls, are clicked.

Note
This property only has an effect when the form is parented to another control. By default, forms always are brought to the front when clicked.

TForm.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

TForm.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TForm.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TForm.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TForm.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Available In: Visual Client Applications

Specifies the inset shadow for the control.

TForm.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TForm.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Available In: Visual Client Applications

Specifies the outset shadow for the control.

TForm.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

TForm.ScrollBars Property

```
property ScrollBars: TScrollBars
```

Available In: Visual Client Applications

Specifies which scrollbars to show, if any.

Note
Even if this property is set to sbHorizontal, sbVertical, or sbBoth, a scrollbar will only be shown if the size of the contents and/or the child controls of the control exceed the client rectangle for the control.

TForm.ScrollSupport Property

```
property ScrollSupport: TScrollSupport
```

Available In: Visual Client Applications

Specifies the directions in which the control can be scrolled, if any.

Note
This property only applies to scrolling via touch and mouse wheel movements, and does not apply to programmatic scrolling or scrolling via the scroll bars themselves.

TForm.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TForm.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TForm.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TForm.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TForm.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TForm.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TForm.OnClientSize Event

```
property OnClientSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the width and/or height of the client area of the control is changed.

TForm.OnClose Event

```
property OnClose: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the form's Close method is called.

TForm.OnCloseQuery Event

```
property OnCloseQuery: TCloseQueryEvent
```

Available In: Visual Client Applications

This event is triggered when the form's Close method is called.

Return True to allow the close to continue, or False to prevent the form from closing.

TForm.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TForm.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TForm.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TForm.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and the user presses a key or key combination.

TForm.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and presses/releases a key or key combination.

TForm.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and the user releases a key or key combination.

TForm.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TForm.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TForm.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TForm.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TForm.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TForm.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

Available In: Visual Client Applications

This event is triggered whenever the mouse wheel is rotated forward or backward.

TForm.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TForm.OnScroll Event

```
property OnScroll: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever a scrollable control is scrolled horizontally or vertically.

TForm.OnScrollSize Event

```
property OnScrollSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the width and/or height of the scrollable area of the control is changed.

TForm.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TForm.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TForm.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TForm.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TForm.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TForm.OnTouchScroll Event

```
property OnTouchScroll: TTouchScrollEvent
```

Available In: Visual Client Applications

This event is triggered whenever a touch moves in any direction over a touch-scroll-enabled control.

TForm.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.84 TFormat Component

Unit: WebUI

Inherits From TElementAttribute

Available In: Visual Client Applications

The TFormat class represents the formatting attributes to use for the content of a UI element or control.

Properties	Methods	Events
Alignment	SetToDefault	
Direction		
Wrap		

TFormat.Alignment Property

```
property Alignment: TContentAlignment
```

Available In: Visual Client Applications

Specifies the alignment of the content.

TFormat.Direction Property

```
property Direction: TContentDirection
```

Available In: Visual Client Applications

Specifies the direction in which the content is displayed.

TFormat.Wrap Property

```
property Wrap: Boolean
```

Available In: Visual Client Applications

Specifies whether the content should be word-wrapped.

TFormat.SetToDefault Method

```
procedure SetToDefault
```

Available In: Visual Client Applications

Use this method to reset the format's properties to their default values.

10.85 TFormatSettings Component

Unit: WebCore

Inherits From TObject

Available In: Client and Server Applications

The TFormatSettings class represents the formatting settings for numeric, date, and time literals. An instance of the TFormatSettings class called **FormatSettings** is automatically created by the component library at application startup, so further instances of the TFormatSettings class should not be created.

Properties	Methods	Events
DateSeparator	Create	
DecimalSeparator		
LongDayNames		
LongMonthNames		
ShortDateFormat		
ShortDateFormatComp		
ShortDayNames		
ShortMonthNames		
ShortTimeFormat		
ShortTimeFormatComp		
StartOfWeek		
TimeAMString		
TimePMString		
TimeSeparator		
Translations		
TwoDigitYearCenturyWindow		

TFormatSettings.DateSeparator Property

```
property DateSeparator: Char
```

Available In: Client and Server Applications

Specifies the character used to separate the various components of a date literal. The default value of this property is the forward slash (/).

Warning
When specifying a ShortDateFormat that uses a different date separator character, please make sure that you modify the DateSeparator property **before** setting the new ShortDateFormat value.

TFormatSettings.DecimalSeparator Property

```
property DecimalSeparator: Char
```

Available In: Client and Server Applications

Specifies the character used to separate the integer portion from the fractional portion in a numeric literal. The default value of this property is the period (.).

TFormatSettings.LongDayNames Property

```
property LongDayNames[Day: Integer]: String
```

Available In: Client and Server Applications

Specifies the full day names for all days of the week. The values are indexed by the day, with Monday being the first at index 1 and Sunday being the last at index 7. The default values are Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, and Sunday.

TFormatSettings.LongMonthNames Property

```
property LongMonthNames[Month: Integer]: String
```

Available In: Client and Server Applications

Specifies the full month names for all months of the year. The values are indexed by the month, with January being the first at index 1 and December being the last at index 12. The default values are January, February, March, April, May, June, July, August, September, October, November, and December.

TFormatSettings.ShortDateFormat Property

```
property ShortDateFormat: String
```

Available In: Client and Server Applications

Specifies the format string used for date literals. The default value of this property is "M/d/yyyy".

The following date format specifiers are supported:

Format Specifier	Description
M	The month number with no leading zero
MM	The month number with a leading zero if the month number is less than 10
d	The day number with no leading zero
dd	The day number with a leading zero if the day number is less than 10
yy	The last two digits of the year number with a leading zero (see the TwoDigitYearCenturyWindow property)
yyyy	The full four digits of the year number

Note
For date/time literals, the ShortDateFormat is used in conjunction with the ShortTimeFormat property, with a space separating the two.

Warning
When specifying a ShortDateFormat that uses a different date separator character, please make sure that you modify the DateSeparator property **before** setting the new ShortDateFormat value.

TFormatSettings.ShortDateFormatComp Property

```
property ShortDateFormatComp[Index: Integer]: String
```

Available In: Client and Server Applications

Accesses a specific component of the short date format by its index in the defined components of the format.

TFormatSettings.ShortDayNames Property

```
property ShortDayNames[Day: Integer]: String
```

Available In: Client and Server Applications

Specifies the abbreviated day names for all days of the week. The values are indexed by the day, with Monday being the first at index 1 and Sunday being the last at index 7. The default values are Mon, Tue, Wed, Thu, Fri, Sat, and Sun.

TFormatSettings.ShortMonthNames Property

```
property ShortMonthNames[Month: Integer]: String
```

Available In: Client and Server Applications

Specifies the abbreviated month names for all months of the year. The values are indexed by the month, with January being the first at index 1 and December being the last at index 12. The default values are Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, and Dec.

TFormatSettings.ShortTimeFormat Property

```
property ShortTimeFormat: String
```

Available In: Client and Server Applications

Specifies the format string used for time literals. The default value of this property is "hh:mm tt".

The following date format specifiers are supported:

Format Specifier	Description
h	The hour number (12-hour clock) with no leading zero
hh	The hour number (12-hour clock) with a leading zero if the hour number is less than 10
H	The hour number (24-hour clock) with no leading zero
HH	The hour number (24-hour clock) with a leading zero if the hour number is less than 10
m	The minute number with no leading zero
mm	The minute number with a leading zero if the minute number is less than 10
s	The second number with no leading zero
ss	The second number with a leading zero if the second number is less than 10
tt	The AM/PM designation for a 12-hour clock literal

Note
For date/time literals, the ShortDateFormat is used in conjunction with the ShortTimeFormat property, with a space separating the two.

Warning
When specifying a ShortTimeFormat that uses a different time separator character, please make sure that you modify the TimeSeparator property **before** setting the new ShortTimeFormat value.

TFormatSettings.ShortTimeFormatComp Property

```
property ShortTimeFormatComp[Index: Integer]: String
```

Available In: Client and Server Applications

Accesses a specific component of the short time format by its index in the defined components of the format.

TFormatSettings.StartOfWeek Property

```
property StartOfWeek: Integer
```

Available In: Client and Server Applications

Specifies the week day number designated as the start of the week for calendar display purposes. The default value is 7, for Sunday, and the valid values are 1 (Monday) through 7 (Sunday).

TFormatSettings.TimeAMString Property

```
property TimeAMString: String
```

Available In: Client and Server Applications

Specifies the literal used to represent the AM designation for a 12-hour clock time literal. The default value of this property is "AM".

TFormatSettings.TimePMString Property

```
property TimePMString: String
```

Available In: Client and Server Applications

Specifies the literal used to represent the PM designation for a 12-hour clock time literal. The default value of this property is "PM".

TFormatSettings.TimeSeparator Property

```
property TimeSeparator: Char
```

Available In: Client and Server Applications

Specifies the character used to separate the various components of a time literal. The default value of this property is the colon (:).

Warning
When specifying a ShortTimeFormat that uses a different time separator character, please make sure that you modify the TimeSeparator property **before** setting the new ShortTimeFormat value.

TFormatSettings.Translations Property

```
property Translations: TStringList
```

Available In: Client and Server Applications

Specifies the translations for all strings used in components/controls and error/warning/information messages as key-value pairs. In order to translate one or more strings, simply reference the string ID using the Values property and assign it a new value. For example, the following will translate the text used for the OK button on message dialogs to a new value:

```
FormatSettings.Translations.Values['DLG_BTN_OK']:='Okey-Dokey';
```

The following strings are pre-defined in the framework and can be translated to suit your needs:

String	Default Value
DLG_MSG	Message
DLG_BTN_OK	OK
DLG_BTN_CANCEL	Cancel
DLG_BTN_ABORT	Abort
DLG_BTN_RETRY	Retry
DLG_BTN_IGNORE	Ignore
DLG_BTN_YES	Yes
DLG_BTN_NO	No
DLG_BTN_ALL	All
DLG_BTN_NOTOALL	No to All
DLG_BTN_YESTOALL	Yes to All
DLG_BTN_CLOSE	Close
TYPE_UNKNOWN	Unknown
TYPE_STRING	String
TYPE_BOOLEAN	Boolean
TYPE_INTEGER	Integer
TYPE_FLOAT	Float
TYPE_DATE	Date
TYPE_TIME	Time
TYPE_DATETIME	DateTime
TYPE_BLOB	Blob
TYPE_SYMBOL	Symbol
ERR_COMP_DESTROY	"%s" component already destroyed

ERR_LOAD_PERSISTENT	Persistent load error (%s)
ERR_LOAD_METHOD	Method %s not found
ERR_SAVE_PERSISTENT	Persistent save error (%s)
ERR_SET_RANGE	Value "%s" out of range for set
ERR_BOOLEAN_LITERAL	Invalid boolean literal "%s" specified
ERR_FORMAT	Error in the format string %s (%s)
ERR_VALUE_CONVERT	Error converting %s value to %s value
ERR_VALUE_READONLY	"%s" is read-only and cannot be modified
ERR_DATETIME_DATE	date
ERR_DATETIME_TIME	time
ERR_DATETIME_MONTH	Invalid month %s specified
ERR_DATETIME_DAY	Invalid day %s specified
ERR_DATETIME_TOOMANYCOMPS	Too many %s components
ERR_DATETIME_MISSINGCOMP	Missing %s component
ERR_DATETIME_INVALIDCOMP	Invalid %s component
ERR_DATETIME_INVALIDFORMAT	Invalid %s format
ERR_DATETIME_INVALID	Invalid %s (%s)
ERR_PARSE_TERMSTR	Unterminated string at %s
ERR_PARSE_MISSING	Missing %s
ERR_PARSE_EXPECT	Expected %s, instead found %s at %s
ERR_LIST_BOUNDS	List index %s out of bounds
ERR_LIST_SORT	You can only use the Find method when a list is sorted
ERR_OWNER	Invalid owner class %s passed to constructor
ERR_LOADUI_STATE	Error loading interface state (%s)
ERR_UI_DUPSTATE	The interface state %s already exists
ERR_UI_ELEMENTCLASS	Cannot find registered element class information for the %s class
ERR_DOM_EVENTADD	Cannot add event handler to "%s" element for "%s" event
ERR_DOM_EVENTCLEAR	Cannot remove "%s" event handler from "%s"
ERR_HTML_FORM_REQUEST	There is no server request assigned to the HTML form
ERR_HTTP_REQUEST_EXECUTING	Request execution already in progress
ERR_HTTP_REQUEST_URL	A URL must be specified for the request
ERR_HTTP_REQUEST_TIMEOUT	Timed out
ERR_HTTP_REQUEST_NETWORK	Network error
ERR_HTTP_REQUEST	Error executing request "%s" (%s)
ERR_AUTH_CANCELLED	The session authentication was cancelled
ERR_MAILER	Error sending mail (%s)
ERR_ROWSET_ROWNOTFOUND	The row was not found in the rowset (ID: %s)

ERR_ROWSET_FINDNEAR	You can only search for nearest matches in the "%s" dataset when searching on columns that match the current sort order
ERR_DATA_TRANSOPTYPE	Invalid or unknown operation type %s specified in the transaction operations
ERR_DATA_DUPCOL	The "%s" column already exists
ERR_DATA_COLNAME	Column names cannot be blank (%s)
ERR_DATA_COLTYPE	Unknown column type (%s)
ERR_DATA_COLLENGTH	Invalid "%s" column length %s
ERR_DATA_COLSCALE	Invalid "%s" column scale %s
ERR_DATA_CONNECT	Cannot connect to server
ERR_DATA_LOADCODE	Status code %s
ERR_DATA_LOAD	Dataset load response error (%s)
ERR_DATA_COMMIT	Database commit response error (%s)
ERR_DATA_TRANSACTIVE	A transaction is not active
ERR_DATA_PENDREQUEST	There are no pending requests
ERR_DATA_COLUMNS	At least one column must be defined for the "%s" dataset
ERR_DATA_OPEN	The "%s" dataset must be open in order to complete this operation
ERR_DATA_NOTOPEN	The "%s" dataset cannot be open when completing this operation
ERR_DATA_NOTEDITING	The "%s" dataset must be in an editable mode before a column can be assigned a value
ERR_DATA_COLNOTFOUND	Column "%s" not found
ERR_DATA_DATASETDB	The "%s" dataset cannot be loaded using the "%s" database
ERR_APP_ERRORTITLE	Application Error
APP_LOAD_MESSAGE	Loading %s...
ERR_DLG_BUTTONS	You must specify at least one button for the message dialog
ERR_FORM_SHOWMODAL	You cannot call ShowModal for the embedded form %s
ERR_CTRL_PARENT	The %s control cannot be a parent of the %s control
ERR_CALENDAR_COLINDEX	Column index %s out of bounds
ERR_CALENDAR_ROWINDEX	Row index %s out of bounds
ERR_GRID_COLINDEX	Column index %s out of bounds
ERR_GRID_ROWINDEX	Row index %s out of bounds
ERR_GRID_COLNOTFOUND	Column "%s" not found
ERR_CANVAS	Your browser does not have HTML5 canvas support
ERR_STORAGE	Your browser does not have HTML5 persistent storage support
ERR_SCRIPT_LOAD	Your browser does not support dynamic script loading
ERR_GEOLOCATION	Your browser does not have HTML5 geolocation support
ERR_MEDIA	Your browser does not have HTML5 media support
ERR_SELECTION	Your browser does not have HTML5 content selection support

ERR_MAP	The map API has not been loaded
ERR_MAP_GEOCODE	Geocoding request error "%s"
ERR_MAP_LOCNOTFOUND	Location "%s" not found
ERR_MAP_DUPLOC	The "%s" location already exists
ERR_MAP_LOCNAME	Location names cannot be blank (%s)
ERR_SIZER_CONTROL	The sizer control itself cannot be assigned as the target control
ERR_ZOOM_FACTOR	Zoom factor %s invalid, factor must be between 1 and 100
ERR_SLIDE_COUNT	At least %s slide images must be specifed before the slide show can be started
ERR_MENU_PARENT	The parent menu "%s" cannot be assigned as a sub-menu

Note
Be sure to include the %s string placeholders in the same order and quantity to ensure that data that is formatted into the string by the runtime still produces a string that makes sense.

TFormatSettings.TwoDigitYearCenturyWindow Property

```
property TwoDigitYearCenturyWindow: Integer
```

Available In: Client and Server Applications

Specifies what century is added to two-digit years when date literals are converted to DateTime values. This property is subtracted from the current year to determine the pivot year. Two digit years that are prior to the pivot year are interpreted as falling in the next century. The default value is 50.

TFormatSettings.Create Method

```
constructor Create
```

Available In: Client and Server Applications

Use this method to create a new instance of the TFormatSettings class.

10.86 TFormControl Component

Unit: WebForms

Inherits From TScrollableControl

Available In: Visual Client Applications

The TFormControl control is the base class for forms, and contains all of the core form functionality in the form of public methods and protected properties/events that descendant classes can use to create customized forms.

Properties	Methods	Events
ModalResult	Close	OnCreate
TabOrder	ShowModal	OnDestroy

TFormControl.ModalResult Property

```
property ModalResult: TModalResult
```

Available In: Visual Client Applications

Specifies the result for a modal form. Assigning a value other than mrNone to this property for a modal form will cause the form to attempt to close. This property has no effect upon forms that were not shown modally.

TFormControl.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the form in the tabbing order for the form's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TFormControl.Close Method

```
procedure Close
```

Available In: Visual Client Applications

Use this method to close the form. When this method is called, the OnCloseQuery event is triggered, followed by the OnClose event. If the OnCloseQuery event handler returns True, then the form will be hidden before the OnClose event is triggered.

TFormControl.ShowModal Method

```
procedure ShowModal
```

Available In: Visual Client Applications

Use this method to show a form in a modal fashion. When a form is shown modally, a modal overlay is placed over all other forms and the entire application surface, and keyboard and mouse input is only allowed for the modal form.

TFormControl.OnCreate Event

```
property OnCreate: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered after the form is created and initialized.

Note
Any design-time components placed on the form will already be instantiated and initialized before this event is triggered.

TFormControl.OnDestroy Event

```
property OnDestroy: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered before the form is destroyed. Use this event to dispose of any instances or resources that may have been allocated in the OnCreate event handler.

10.87 TFormElement Component

Unit: WebUI

Inherits From TElement

Available In: Visual Client Applications

The TFormElement class is the element class for HTML form elements, and contains all of the HTML form functionality in the form of public methods and properties/events that control classes can use to create HTML form controls.

> **Note**
> This element does not provide support for HTML forms at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events
Action	Reset	
Encoding	Submit	
Method		
Target		

TFormElement.Action Property

```
property Action: String
```

Available In: Visual Client Applications

Specifies the URL to use for the form submittal when the Submit method is called. The default value is ''.

TFormElement.Encoding Property

```
property Encoding: THTMLFormEncoding
```

Available In: Visual Client Applications

Specifies the type of MIME content encoding for the form data that is sent when the Submit method is called.

TFormElement.Method Property

```
property Method: THTMLFormMethod
```

Available In: Visual Client Applications

Specifies the HTTP method used for the form submittal when the Submit method is called.

TFormElement.Target Property

```
property Target: String
```

Available In: Visual Client Applications

Specifies the name of a TFrameElement that should accept all output from the form submittal request to the web server when the Submit method is called. The default value is ''.

TFormElement.Reset Method

```
procedure Reset
```

Available In: Visual Client Applications

Use this method to clear all form input elements contained within the form element.

TFormElement.Submit Method

```
procedure Submit
```

Available In: Visual Client Applications

Use this method to submit all form input elements contained within the form element to the web server using the URL specified by the Action property, the MIME content type encoding specified by the Encoding property, and the HTTP method specifed by the Method property.

10.88 TFrame Component

Unit: WebForms

Inherits From TFormControl

Available In: Visual Client Applications

The TFrame component represents a frame control, which is a basic, non-scrollable version of a form control that is useful for situations where you want auto-sizing capabilities and don't need the scrolling features of a regular form control. Typically, this is when you wish to embed frame instances in another parent control.

Please see the Creating and Showing Forms for more information on using forms.

Properties	Methods	Events
ActivateOnClick		OnAnimationComplete
AutoSize		OnAnimationsComplete
Background		OnCaptureEnd
Border		OnCaptureStart
Corners		OnCapturing
Cursor		OnClick
InsetShadow		OnClientSize
Opacity		OnClose
OutsetShadow		OnCloseQuery
Padding		OnContextMenu
		OnDblClick
		OnHide
		OnKeyDown
		OnKeyPress
		OnKeyUp
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMouseWheel
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd

		OnTouchMove
		OnTouchStart

TFrame.ActivateOnClick Property

```
property ActivateOnClick: Boolean
```

Available In: Visual Client Applications

Specifies whether the form should automatically be brought to the front when it, or any child controls, are clicked.

Note
This property only has an effect when the form is parented to another control. By default, forms always are brought to the front when clicked.

TFrame.AutoSize Property

```
property AutoSize: TAutoSize
```

Available In: Visual Client Applications

Specifies how (if at all) the control should automatically be sized based upon the child controls placed in the frame.

TFrame.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

TFrame.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TFrame.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TFrame.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TFrame.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Available In: Visual Client Applications

Specifies the inset shadow for the control.

TFrame.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TFrame.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Available In: Visual Client Applications

Specifies the outset shadow for the control.

TFrame.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

TFrame.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TFrame.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TFrame.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TFrame.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TFrame.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TFrame.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TFrame.OnClientSize Event

```
property OnClientSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the width and/or height of the client area of the control is changed.

TFrame.OnClose Event

```
property OnClose: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the form's Close method is called.

TFrame.OnCloseQuery Event

```
property OnCloseQuery: TCloseQueryEvent
```

Available In: Visual Client Applications

This event is triggered when the form's Close method is called.

Return True to allow the close to continue, or False to prevent the form from closing.

TFrame.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TFrame.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TFrame.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TFrame.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and the user presses a key or key combination.

TFrame.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and presses/releases a key or key combination.

TFrame.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and the user releases a key or key combination.

TFrame.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TFrame.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TFrame.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TFrame.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TFrame.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TFrame.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

Available In: Visual Client Applications

This event is triggered whenever the mouse wheel is rotated forward or backward.

TFrame.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TFrame.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TFrame.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TFrame.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TFrame.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TFrame.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TFrame.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.89 TFrameElement Component

Unit: WebUI

Inherits From TWebElement

Available In: Visual Client Applications

The TFrameElement class is the element class for embedded iframe elements, and contains all of the iframe functionality in the form of public methods and properties/events that control classes can use to create iframe controls.

> **Note**
> This element does not provide support for iframes at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events
Document	Print	
DocumentText		
Scrolling		
TargetName		
Window		

TFrameElement.Document Property

```
property Document: THTMLDocument
```

Available In: Visual Client Applications

Returns the DOM (Document Object Model) document instance of the currently-loaded HTML document. If the URL property has been specified, then this property will return the document instance once the OnLoad event has been triggered and the Loaded property is True.

Note
Accessing the DOM document instance allows you to manipulate the children of the DOM document instance in code instead of having to use HTML strings, which is the case when using the DocumentText property. However, this access is subject to same-origin security constraints, and will be denied if the contents of the frame element were loaded from a different origin.

TFrameElement.DocumentText Property

```
property DocumentText: String
```

Available In: Visual Client Applications

Returns the currently-loaded HTML document in the element as a string. If the URL property has been specified, then this property will return the document contents once the OnLoad event has been triggered and the Loaded property is True.

Note
You can also assign a valid HTML string to this property, in which case the URL property is automatically cleared.

TFrameElement.Scrolling Property

```
property Scrolling: Boolean
```

Available In: Visual Client Applications

Specifies whether scrolling should be enabled for the element.

Note
This property is required because certain browsers require a special way of specifying whether scrollbars should be shown for an embedded iframe element.

TFrameElement.TargetName Property

```
property TargetName: String
```

Available In: Visual Client Applications

Specifies the name of the embedded iframe element, which is required in order to allow output of actions like HTML form submittals to be redirected to a specific iframe element.

TFrameElement.Window Property

```
property Window: TWindow
```

Available In: Visual Client Applications

Returns the DOM (Document Object Model) window instance of the frame element.

Note
Accessing the DOM window instance allows you to make calls into the global execution environment of the frame element. However, this access is subject to same-origin security constraints, and will be denied if the contents of the frame element were loaded from a different origin.

TFrameElement.Print Method

```
procedure Print
```

Available In: Visual Client Applications

Use this method to print the currently-loaded HTML document in the element. If no HTML document is loaded, then this method does nothing.

10.90 TGradient Component

Unit: WebUI

Inherits From TElementAttribute

Available In: Visual Client Applications

The TGradient class represents a background gradient for a UI element or control. Background gradients can be linear or radial, and are comprised of 2 or more color stops.

Properties	Methods	Events
Angle	SetToDefault	
AutoCenter		
CenterX		
CenterY		
ColorStops		
GradientType		

TGradient.Angle Property

```
property Angle: Integer
```

Available In: Visual Client Applications

Specifies the angle of the linear gradient in degrees. The angle is the direction in which the linear gradient will transition from the first color stop to the last color stop, with both 0 degrees and 360 degrees being the top of the background area of a UI element or control. The gradient is a linear gradient when the GradientType property is set to gtLinear.

TGradient.AutoCenter Property

```
property AutoCenter: Boolean
```

Available In: Visual Client Applications

Specifies whether the radial gradient should be auto-centered in the exact middle of the background area of a UI element or control. The gradient is a radial gradient when the GradientType property is set to gtRadial.

TGradient.CenterX Property

```
property CenterX: Integer
```

Available In: Visual Client Applications

Specifies the horizontal position of the center of the radial gradient, relative to the background area of a UI element or control. The gradient is a radial gradient when the GradientType property is set to gtRadial.

TGradient.CenterY Property

```
property CenterY: Integer
```

Available In: Visual Client Applications

Specifies the vertical position of the center of the radial gradient, relative to the background area of a UI element or control. The gradient is a radial gradient when the GradientType property is set to gtRadial.

TGradient.ColorStops Property

```
property ColorStops: TGradientColorStops
```

Available In: Visual Client Applications

Specifies the color stops for the gradient. Each color stop is represented by a color and a position (0-100), with each color transitioning to the next at the specified positions. Each position is relative to the existing width and height of the background area for a UI element or control, and adjusts proportionally as the background area is resized.

> **Note**
> A gradient always requires at least 2 color stops at positions 0 and 100.

TGradient.GradientType Property

```
property GradientType: TGradientType
```

Available In: Visual Client Applications

Specifies the type of gradient, linear or radial.

TGradient.SetToDefault Method

```
procedure SetToDefault
```

Available In: Visual Client Applications

Use this method to reset the gradient's properties to their default values.

10.91 TGradientColorStop Component

Unit: WebUI

Inherits From TElementAttribute

Available In: Visual Client Applications

The TGradientColorStop class represents a background gradient color stop for a UI element or control.

Properties	Methods	Events
Color	SetToDefault	
Position		

TGradientColorStop.Color Property

```
property Color: TColor
```

Available In: Visual Client Applications

Specifies the color for the gradient color stop.

TGradientColorStop.Position Property

```
property Position: Integer
```

Available In: Visual Client Applications

Specifies the relative position for the gradient color stop.

TGradientColorStop.SetToDefault Method

```
procedure SetToDefault
```

Available In: Visual Client Applications

Use this method to reset the gradient color stop's properties to their default values.

10.92 TGradientColorStops Component

Unit: WebUI

Inherits From TElementAttribute

Available In: Visual Client Applications

The TGradientColorStops class represents the background gradient color stops for a UI element or control.

Properties	Methods	Events
ColorStop	Add	
Count	Remove	
	RemoveAll	
	SetToDefault	

TGradientColorStops.ColorStop Property

```
property ColorStop[Index: Integer]: TGradientColorStop
```

Available In: Visual Client Applications

Accesses a gradient color stop by its index position (0 to Count property).

TGradientColorStops.Count Property

```
property Count: Integer
```

Available In: Visual Client Applications

Indicates the number of defined gradient color stops.

TGradientColorStops.Add Method

```
function Add: TGradientColorStop
```

Available In: Visual Client Applications

Use this method to add a new gradient color stop. The color stop will be initialized with a Color value of clBlack and a Position value of 0.

TGradientColorStops.Remove Method

```
procedure Remove(Index: Integer)
```

Available In: Visual Client Applications

Use this method to remove an existing gradient color stop by its index position in the color stops.

TGradientColorStops.RemoveAll Method

```
procedure RemoveAll
```

Available In: Visual Client Applications

Use this method to remove all existing gradient color stops.

TGradientColorStops.SetToDefault Method

```
procedure SetToDefault
```

Available In: Visual Client Applications

Use this method to reset the gradient color stops' properties to their default values.

10.93 TGrid Component

Unit: WebGrids

Inherits From TGridControl

Available In: Visual Client Applications

The TGrid component represents a grid control. A grid can be un-bound and used to display and update a matrix of cells as strings, or can be bound to TDataSet component instances in order to display and update the rows in the dataset.

Each grid instance has its own defined set of columns and, if the grid is bound to a dataset, each column can be bound to a specific dataset column. Grid columns can also be used to sort a dataset by a specific set of columns via their SortDirection property.

Properties	Methods	Events
AllowAppends		OnAnimationComplete
AllowDeletes		OnAnimationsComplete
AllowInserts		OnCaptureEnd
Background		OnCaptureStart
Border		OnCapturing
ColumnHeaders		OnClick
ColumnHeadersHeight		OnColumnChanged
Corners		OnContextMenu
Cursor		OnDblClick
DataSet		OnHide
Enabled		OnKeyDown
Hint		OnKeyPress
MultiSelect		OnKeyUp
ReadOnly		OnMouseDown
RowHeight		OnMouseEnter
RowSelect		OnMouseLeave
ScrollBars		OnMouseMove
ScrollSupport		OnMouseUp
ShowLines		OnMouseWheel
TabOrder		OnMove
TabStop		OnRowChanged
WantTabs		OnScroll
		OnShow
		OnSize
		OnTouchCancel

		OnTouchEnd
		OnTouchMove
		OnTouchScroll
		OnTouchStart

TGrid.AllowAppends Property

```
property AllowAppends: Boolean
```

Available In: Visual Client Applications

Specifies whether the grid should allow new rows to be appended by the user moving the active row index past the last row in the grid. The default value is True.

TGrid.AllowDeletes Property

```
property AllowDeletes: Boolean
```

Available In: Visual Client Applications

Specifies whether the grid should allow rows to be deleted by the user using the Ctrl-Delete key combination in the grid. The default value is True.

TGrid.AllowInserts Property

```
property AllowInserts: Boolean
```

Available In: Visual Client Applications

Specifies whether the grid should allow new rows to be inserted by the user using the Insert or Shift-Insert key combination in the grid. The default value is True.

TGrid.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background for the control.

TGrid.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TGrid.ColumnHeaders Property

```
property ColumnHeaders: Boolean
```

Available In: Visual Client Applications

Specifies whether column headers should be shown for the columns in the grid.

TGrid.ColumnHeadersHeight Property

```
property ColumnHeadersHeight: Integer
```

Available In: Visual Client Applications

Specifies the height of the column headers in the grid.

TGrid.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TGrid.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TGrid.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the grid to. The default value is nil.

Note
In order to actually show data from the dataset, the grid Columns must also have their DataColumn property set to valid dataset column names.

TGrid.Enabled Property

```
property Enabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TGrid.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TGrid.MultiSelect Property

```
property MultiSelect: Boolean
```

Available In: Visual Client Applications

Specifies whether multiple rows can be selected in the grid. The SelectedCount property can be examined to find out how many rows are selected and the Selected property can be examined to find out which rows are selected.

Note
You can only use the multi-select functionality when the RowSelect property is True. Setting the RowSelect property to False will automatically set the MultiSelect property to False also.

TGrid.ReadOnly Property

```
property ReadOnly: Boolean
```

Available In: Visual Client Applications

Specifies whether the grid's rows and columns can be modified by the user. The default value is False.

Note
The grid rows and columns can always be programmatically modified.

TGrid.RowHeight Property

```
property RowHeight: Integer
```

Available In: Visual Client Applications

Indicates the height, in pixels, of each visible row in the grid.

TGrid.RowSelect Property

```
property RowSelect: Boolean
```

Available In: Visual Client Applications

Specifies that the active selection should always encompass the entire current row. The default value is False.

TGrid.ScrollBars Property

```
property ScrollBars: TScrollBars
```

Available In: Visual Client Applications

Specifies which scrollbars to show, if any.

Note
Even if this property is set to sbHorizontal, sbVertical, or sbBoth, a scrollbar will only be shown if the size of the contents of the control exceed the client rectangle for the control.

TGrid.ScrollSupport Property

```
property ScrollSupport: TScrollSupport
```

Available In: Visual Client Applications

Specifies the directions in which the control can be scrolled, if any.

> **Note**
> This property only applies to scrolling via touch and mouse wheel movements, and does not apply to programmatic scrolling or scrolling via the scroll bars themselves.

TGrid.ShowLines Property

```
property ShowLines: Boolean
```

Available In: Visual Client Applications

Specifies that horizontal and vertical lines should be used to separate the various grid columns and rows.

TGrid.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TGrid.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TGrid.WantTabs Property

```
property WantTabs: Boolean
```

Available In: Visual Client Applications

Specifies that the tab key can be used to navigate from cell to cell in the grid. The default value is True.

Note
The RowSelect property must be False for this property setting to take effect. If the RowSelect property is True, then the tab key will cause the input focus to move to the next control within the parent container control for the grid.

TGrid.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TGrid.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TGrid.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TGrid.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TGrid.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TGrid.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TGrid.OnColumnChanged Event

```
property OnColumnChanged: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the ColumnIndex property changes.

TGrid.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TGrid.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TGrid.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TGrid.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses a key or key combination.

TGrid.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TGrid.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user releases a key or key combination.

TGrid.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TGrid.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TGrid.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TGrid.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TGrid.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TGrid.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

Available In: Visual Client Applications

This event is triggered whenever the mouse wheel is rotated forward or backward.

TGrid.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TGrid.OnRowChanged Event

```
property OnRowChanged: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the RowIndex property changes.

TGrid.OnScroll Event

```
property OnScroll: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever a scrollable control is scrolled horizontally or vertically.

TGrid.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TGrid.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TGrid.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TGrid.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TGrid.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TGrid.OnTouchScroll Event

```
property OnTouchScroll: TTouchScrollEvent
```

Available In: Visual Client Applications

This event is triggered whenever a touch moves in any direction over a touch-scroll-enabled control.

TGrid.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.94 TGridCell Component

Unit: WebGrids

Inherits From TControl

Available In: Visual Client Applications

The TGridCell component represents a visible grid cell. A grid cell is used to display the values for a given grid column in a grid control. The Background and Font properties can be modified in a TGridColumn OnCellUpdate event handler to affect how data is displayed in the grid.

Properties	Methods	Events
Background		
Border		
Data		
Font		
Hint		
Index		

TGridCell.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the cell.

TGridCell.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the cell.

TGridCell.Data Property

```
property Data: String
```

Available In: Visual Client Applications

Indicates the current contents of the cell.

TGridCell.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the cell.

TGridCell.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the cell for a browser-specific amount of time. The default value is ''.

TGridCell.Index Property

```
property Index: Integer
```

Available In: Visual Client Applications

Indicates the index of the cell in the grid column.

10.95 TGridColumn Component

Unit: WebGrids

Inherits From TBindableColumnControl

Available In: Visual Client Applications

The TGridColumn component represents a grid column. A grid column can be un-bound and used to display and update a vertical list of cells as strings, or can be bound to one of the TDataSet Columns in order to display and update the column values for rows in the dataset.

Grid columns can be used to sort a dataset by a specific set of columns via their SortDirection property.

The ControlType property controls what type of control, if any, will be used to edit the contents of the cells for the grid column.

Properties	Methods	Events
Alignment	HideDropDown	OnButtonClick
AllowSize	ShowDropDown	OnCellUpdate
AutoDropDown	ToggleSortDirection	OnCompare
CalendarDefaultView		OnDropDownHide
CalendarHeight		OnDropDownShow
CalendarWidth		OnHeaderClick
ControlType		OnHide
Cursor		OnShow
DataColumn		OnSize
Direction		OnUpdate
DropDownItemCount		
DropDownPosition		
DropDownVisible		
Enabled		
Font		
Header		
ImageLayout		
Index		
Items		
ItemsSorted		
MaxLength		
ParentGrid		
ReadOnly		
SingleClickToggle		

SortDirection		
SortIndex		
SpellCheck		
StretchToFit		
ValueSelected		
ValueUnselected		
Wrap		

TGridColumn.Alignment Property

```
property Alignment: TContentAlignment
```

Available In: Visual Client Applications

Specifies the alignment of the cell text for the grid column.

TGridColumn.AllowSize Property

```
property AllowSize: Boolean
```

Available In: Visual Client Applications

Specifies whether the grid column can be sized by moving the mouse over the right border of the grid column header. The default value is True.

TGridColumn.AutoDropDown Property

```
property AutoDropDown: Boolean
```

Available In: Visual Client Applications

Specifies that the drop-down list should automatically be shown when the user starts typing in the grid column's edit control. This property is only applicable when the ControlType property is set to ctEditComboBox.

TGridColumn.CalendarDefaultView Property

```
property CalendarDefaultView: TCalendarView
```

Available In: Visual Client Applications

Specifies the default view for the drop-down calendar control when the ControlType property is set to ctDateEditCombobBox. The default view determines both the initial view shown in the calendar after it is created, as well as the minimum view that the user is permitted to navigate to. The default value is cvMonth.

TGridColumn.CalendarHeight Property

```
property CalendarHeight: Integer
```

Available In: Visual Client Applications

Specifies the height of the drop-down calendar control when the ControlType property is set to ctDateEditCombobBox.

TGridColumn.CalendarWidth Property

```
property CalendarWidth: Integer
```

Available In: Visual Client Applications

Specifies the width of the drop-down calendar control when the ControlType property is set to ctDateEditCombobBox.

TGridColumn.ControlType Property

```
property ControlType: TGridColumnControlType
```

Available In: Visual Client Applications

Specifies the control type to use for any in-place editing for the column. The default value is ctNone, which means that the column cannot be edited.

If the ControlType is set to ctLink, then the cell or bound dataset values should be in the format of:

```
<URL> [; <Optional Title>]
```

If the ControlType is set to ctImage, then the cell or bound dataset values should be in the format of:

```
<URL> [; <Optional Hint>]
```

Note
If the <URL> specified in the cell or bound dataset value for an image column (ControlType=ctImage) contains a data-URL value (for example, ...), then you cannot specify an optional hint for the image.

TGridColumn.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TGridColumn.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TGridColumn.Direction Property

```
property Direction: TContentDirection
```

Available In: Visual Client Applications

Specifies the direction in which the grid column's content is displayed/edited.

TGridColumn.DropDownItemCount Property

```
property DropDownItemCount: Integer
```

Available In: Visual Client Applications

Specifies the number of visible items to display in the drop-down list of the grid column's edit control when the ControlType property is set to ctEditComboBox.

TGridColumn.DropDownPosition Property

```
property DropDownPosition: TDropDownPosition
```

Available In: Visual Client Applications

Specifies the position of the drop-down list/calendar of the grid column's edit control when the ControlType property is set to ctEditComboBox or ctDateEditCombobBox.

TGridColumn.DropDownVisible Property

```
property DropDownVisible: Boolean
```

Available In: Visual Client Applications

Indicates whether the drop-down list of the grid column's edit control is visible when the ControlType property is set to ctEditComboBox or ctDialogEditComboBox.

TGridColumn.Enabled Property

```
property Enabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the column is enabled or disabled. When a grid column is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TGridColumn.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TGridColumn.Header Property

```
property Header: TGridHeader
```

Available In: Visual Client Applications

Specifies the header properties for the grid column. The grid column header is only visible when the TGrid ColumnHeaders property is set to True.

TGridColumn.ImageLayout Property

```
property ImageLayout: TContentLayout
```

Available In: Visual Client Applications

Specifies the layout properties of the image content contained within the column when the ControlType property is set to ctImage.

TGridColumn.Index Property

```
property Index: Integer
```

Available In: Visual Client Applications

Indicates the index of the column in the list of grid columns.

TGridColumn.Items Property

```
property Items: TStrings
```

Available In: Visual Client Applications

Specifies the list of items to use in the drop-down list of the grid column's edit control when the ControlType property is set to ctEditComboBox.

TGridColumn.ItemsSorted Property

```
property ItemsSorted: Boolean
```

Available In: Visual Client Applications

Specifies whether the list of items to display in the drop-down list of the grid column's edit control are sorted when the ControlType property is set to ctEditComboBox.

TGridColumn.MaxLength Property

```
property MaxLength: Integer
```

Available In: Visual Client Applications

Specifies the maximum allowable length for any text in the grid column's edit control when the ControlType property is set to ctEdit, ctEditComboBox, ctDialogEditComboBox, or ctMultiLineEdit.

TGridColumn.ParentGrid Property

```
property ParentGrid: TGridControl
```

Available In: Visual Client Applications

Indicates the parent grid control that contains the column, or nil if the column has not been assigned to a grid control.

TGridColumn.ReadOnly Property

```
property ReadOnly: Boolean
```

Available In: Visual Client Applications

Specifies whether or not the grid column can be modified.

TGridColumn.SingleClickToggle Property

```
property SingleClickToggle: Boolean
```

Available In: Visual Client Applications

Specifies that whether a click (True) or double-click (False) is required in order to toggle the selected state of the grid column's control. The default value is False. This property is used when the ControlType property is set to ctCheckBox.

TGridColumn.SortDirection Property

```
property SortDirection: TSortDirection
```

Available In: Visual Client Applications

Specifies the sort for the column. If the grid is data-bound (DataSet property is not nil) and the column has been assigned a valid column name in the DataColumn, then assigning sdAscending or sdDescending to this property will cause the dataset to add the column specified in the DataColumn property to the active sort for the dataset. Assigning sdNone to this property will cause the dataset to remove the column specified in the DataColumn property from the active sort for the dataset.

If the grid is un-bound, then assigning sdAscending or sdDescending to this property will cause the grid to add the column to the active sort. Assigning sdNone to this property will cause the grid to remove the column from the active sort.

The default value is sdNone.

If the ColumnHeaders property is True when this property is changed, then the column header will reflect the sort status of the column.

Note
The TDataSet Sort method is automatically called by the grid when the grid is data-bound and this property is changed. If the grid is un-bound, then the TGrid SortRows method is automatically called when this property is changed.

TGridColumn.SortIndex Property

```
property SortIndex: Integer
```

Available In: Visual Client Applications

If the SortDirection property is not equal to sdNone, then this property indicates the position of the column in the active sort for an un-bound grid.

TGridColumn.SpellCheck Property

```
property SpellCheck: Boolean
```

Available In: Visual Client Applications

Specifies whether spell-checking is enabled for the grid column's edit control when the ControlType property is set to ctEdit, ctEditComboBox, ctDialogEditComboBox, or ctMultiLineEdit.

TGridColumn.StretchToFit Property

```
property StretchToFit: Boolean
```

Available In: Visual Client Applications

Specifies whether the column should automatically stretch to fit against the right edge of the client rectangle for the grid.

TGridColumn.ValueSelected Property

```
property ValueSelected: String
```

Available In: Visual Client Applications

Specifies the textual value to use for the selected state when reading and writing data to and from the DataColumn that the control is bound to. This property is used when the ControlType property is set to ctCheckBox. The default value is "True".

TGridColumn.ValueUnselected Property

```
property ValueUnselected: String
```

Available In: Visual Client Applications

Specifies the textual value to use for the unselected state when reading and writing data to and from the DataColumn that the control is bound to. This property is used when the ControlType property is set to ctCheckBox. The default value is "False".

TGridColumn.Wrap Property

```
property Wrap: Boolean
```

Available In: Visual Client Applications

Specifies whether the text in the column should wrap if it exceeds the width of the column. The default value is False.

TGridColumn.HideDropDown Method

```
procedure HideDropDown
```

Available In: Visual Client Applications

Use this method to hide the drop-down control associated with the column (if visible) when the ControlType property is set to ctEditComboBox, ctDateEditCombobBox, or ctDialogEditComboBox.

TGridColumn.ShowDropDown Method

```
procedure ShowDropDown
```

Available In: Visual Client Applications

Use this method to show the drop-down control associated with the control (it not already visible) when the ControlType property is set to ctEditComboBox, ctDateEditCombobBox, or ctDialogEditComboBox.

TGridColumn.ToggleSortDirection Method

```
procedure ToggleSortDirection(AClear: Boolean=False)
```

Available In: Visual Client Applications

Use this method to toggle the column's SortDirection property.

TGridColumn.OnButtonClick Event

```
property OnButtonClick: TClickEvent
```

Available In: Visual Client Applications

This event is triggered whenever the associated combo button is clicked for any grid column whose ControlType is ctEditComboBox, ctDateEditComboBox, or ctDialogEditComboBox.

Return True to allow the default click behavior and False to prevent the default click behavior from occurring.

TGridColumn.OnCellUpdate Event

```
property OnCellUpdate: TGridColumnCellEvent
```

Available In: Visual Client Applications

This event is triggered whenever the contents or display properties of a cell in the grid column are updated, and can be used to override the default display of the grid cell parameter by modifying its background and/or font.

TGridColumn.OnCompare Event

```
property OnCompare: TGridColumnCompareEvent
```

Available In: Visual Client Applications

This event is triggered whenever the grid column is sorted in an un-bound grid, and can be used to override the default sorting of the string column values.

TGridColumn.OnDropDownHide Event

```
property OnDropDownHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the associated drop-down control is hidden when the ControlType property is set to ctEditComboBox, ctDateEditCombobBox, or ctDialogEditComboBox.

TGridColumn.OnDropDownShow Event

```
property OnDropDownShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the associated drop-down control is shown when the ControlType property is set to ctEditComboBox, ctDateEditCombobBox, or ctDialogEditComboBox.

TGridColumn.OnHeaderClick Event

```
property OnHeaderClick: TGridHeaderClickEvent
```

Available In: Visual Client Applications

This event is triggered when the grid column's header is clicked. Return False from any event handler attached to this event in order to prevent the default action when the header is clicked.

TGridColumn.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TGridColumn.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TGridColumn.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TGridColumn.OnUpdate Event

```
property OnUpdate: TGridColumnUpdateEvent
```

Available In: Visual Client Applications

This event is triggered for un-bound grid controls whenever the contents of a cell in the grid column is updated.

10.96 TGridControl Component

Unit: WebGrids

Inherits From TBindableControl

Available In: Visual Client Applications

The TGridControl control is the base class for grids, and contains all of the core grid functionality in the form of public methods and protected properties/events that descendant classes can use to create customized grids.

Properties	Methods	Events
AlwaysShowControls	AddColumnsFromDataSet	
ColumnControlActive	AppendRow	
ColumnControlVisible	DeleteRow	
ColumnCount	FirstColumn	
ColumnIndex	FirstRow	
Columns	GetGridColumn	
RowCount	GetRows	
RowIndex	HideColumnControl	
RowOffset	InsertRow	
Rows	LastColumn	
RowVisible	LastRow	
Selected	LoadRows	
SelectedCount	MakeColumnVisible	
SortCaseInsensitive	MakeRowVisible	
Sorted	NewColumn	
SortLocaleInsensitive	NextColumn	
VisibleColumnCount	NextPage	
VisibleColumns	NextRow	
	PriorColumn	
	PriorPage	
	PriorRow	
	RemoveColumn	
	ScrollNext	
	ScrollNextPage	
	ScrollPrior	
	ScrollPriorPage	
	ScrollTo	
	SelectAll	

	SelectRange	
	SetToColumn	
	SetToRow	
	ShowColumnControl	
	SortRows	
	ToggleSelected	
	UpdateCells	

TGridControl.AlwaysShowControls Property

```
property AlwaysShowControls: Boolean
```

Available In: Visual Client Applications

Specifies whether the grid should always show any column controls for the grid columns. A grid column has a control associated with it when the TGridColumn ControlType property is not equal to ctNone.

> **Note**
> Not all column controls offer text editing capabilities. Some are used strictly for displaying the data in the grid column, while others like the checkbox control offer editing via double-click interactions.

TGridControl.ColumnControlActive Property

```
property ColumnControlActive: Boolean
```

Available In: Visual Client Applications

Indicates whether any grid column control is currently active. The control for a grid column is controlled via the TGridColumn ControlType property, and the grid column controls can be made visible by calling the ShowColumnControl method. This property can only be true if the ColumnControlVisible property is True.

Note
Even if the ColumnControlVisible property is True, it is possible that there won't be any active column controls because row selection is turned on for the grid control or the column doesn't use a control for display or editing. Use this property to determine if any grid column controls are actually active.

TGridControl.ColumnControlVisible Property

```
property ColumnControlVisible: Boolean
```

Available In: Visual Client Applications

Indicates whether any grid column controls should be visible. The control for a grid column is controlled via the TGridColumn ControlType property, and the grid column controls can be made visible by calling the ShowColumnControl method.

Note
Even if the ColumnControlVisible property is True, it is possible that there won't be any active column controls because row selection is turned on for the grid control or the column doesn't use a control for display or editing. You can use the ColumnControlActive property to determine if any grid column controls are actually active.

TGridControl.ColumnCount Property

```
property ColumnCount: Integer
```

Available In: Visual Client Applications

Indicates the number of columns in the grid control.

TGridControl.ColumnIndex Property

```
property ColumnIndex: Integer
```

Available In: Visual Client Applications

Specifies the current column index for the grid control. If there is no current column, this property will be -1.

TGridControl.Columns Property

```
property Columns[AIndex: Integer]: TGridColumn

property Columns[const AName: String]: TGridColumn
```

Available In: Visual Client Applications

Contains the defined columns for the grid control.

TGridControl.RowCount Property

```
property RowCount: Integer
```

Available In: Visual Client Applications

Indicates the number of rows in the grid control. If the grid control has been bound to a TDataSet instance, then this property will be equal to the RowCount property of the dataset.

TGridControl.RowIndex Property

```
property RowIndex: Integer
```

Available In: Visual Client Applications

Specifies the current row index for the grid control. If there is no current row, this property will be -1.

TGridControl.RowOffset Property

```
property RowOffset: Integer
```

Available In: Visual Client Applications

Specifies the current row offset for the grid control. The row offset is the the index of the row that is the first visible row. If there are now rows present in the grid control, this property will be 0.

TGridControl.Rows Property

```
property Rows: TGridRows
```

Available In: Visual Client Applications

Contains the rows for the grid when the grid control is un-bound.

TGridControl.RowVisible Property

```
property RowVisible: Boolean
```

Available In: Visual Client Applications

Indicates whether the current row, specified by the RowIndex property, is visible.

TGridControl.Selected Property

```
property Selected[ARowIndex: Integer]: Boolean
```

Available In: Visual Client Applications

Accesses the selection state of each row in the grid by its index.

TGridControl.SelectedCount Property

```
property SelectedCount: Integer
```

Available In: Visual Client Applications

Indicates the number of selected rows in the grid.

TGridControl.SortCaseInsensitive Property

```
property SortCaseInsensitive: Boolean
```

Available In: Visual Client Applications

Specifies whether or not an active sort on an un-bound grid, as indicated by the Sorted property, should be case-sensitive.

Note
Changing this property will trigger a sort on an un-bound grid if the Sorted property is True.

TGridControl.Sorted Property

```
property Sorted: Boolean
```

Available In: Visual Client Applications

Indicates whether any of the Columns in the grid control have their SortDirection property set to sdAscending or sdDescending.

Note
This property does **not** indicate whether an un-bound grid control has actually been sorted yet via the Sort method. The sorting is designed this way in order to allow multiple columns to be designated as sort columns without triggering an automatic sort operation each time.

TGridControl.SortLocaleInsensitive Property

```
property SortLocaleInsensitive: Boolean
```

Available In: Visual Client Applications

Specifies whether or not an active sort on an un-bound grid, as indicated by the Sorted property, should be locale-sensitive.

Note
Changing this property will trigger a sort on an un-bound grid if the Sorted property is True.

TGridControl.VisibleColumnCount Property

```
property VisibleColumnCount: Integer
```

Available In: Visual Client Applications

Indicates the number of visible columns in the grid control.

TGridControl.VisibleColumns Property

```
property VisibleColumns[AIndex: Integer]: TGridColumn
```

Available In: Visual Client Applications

Accesses the visible columns in the grid control by their index into the list of visible columns.

TGridControl.AddColumnsFromDataSet Method

```
procedure AddColumnsFromDataSet
```

Available In: Visual Client Applications

Use this method to quickly create grid columns for all defined columns in the dataset referenced in the DataSet property.

> **Note**
> This method does **not** clear out any existing grid columns before creating the new columns.

TGridControl.AppendRow Method

```
procedure AppendRow
```

Available In: Visual Client Applications

Use this method to append a row to the end of any existing rows in the grid control.

TGridControl.DeleteRow Method

```
procedure DeleteRow
```

Available In: Visual Client Applications

Use this method to delete the current row, represented by the RowIndex property, from the grid control.

TGridControl.FirstColumn Method

```
function FirstColumn: Boolean
```

Available In: Visual Client Applications

Use this method to change the current grid column to the first visible column in the grid, if one exists, and update the ColumnIndex property accordingly.

TGridControl.FirstRow Method

```
procedure FirstRow(ShiftKey, CtrlKey: Boolean=False)
```

Available In: Visual Client Applications

Use this method to change the current grid row to the first row in the grid, if one exists, and update the RowIndex property accordingly.

Note
The ShiftKey and CtrlKey parameters control how multiple row selection occurs in relation to the row navigation. When the ShiftKey parameter is True, then the range of selected rows is extended to include all rows between the current row index and the row at the new index. When the CtrlKey parameter is True, then the set of selected rows is extended to include the new row at the new index.

TGridControl.GetGridColumn Method

```
function GetGridColumn(Value: TDataColumn): TGridColumn
```

Available In: Visual Client Applications

Use this method to lookup a grid column by which TDataSet column it is bound to.

TGridControl.GetRows Method

```
function GetRows: String
```

Available In: Visual Client Applications

Use this method to retrieve the rows of an un-bound grid as a JSON string. The JSON is formatted as follows:

```
{   "rows": [
{ "GridColumn1": "Test 1", "GridColumn2": "100", "GridColumn3": "" },
{ "GridColumn1": "Test 2", "GridColumn2": "200", "GridColumn3": "" },
{ "GridColumn1": "Test 3", "GridColumn2": "300", "GridColumn3": "" }
] }
```

TGridControl.HideColumnControl Method

```
procedure HideColumnControl
```

Available In: Visual Client Applications

Use this method to hide any column controls for the columns in the grid. The control for a grid column is controlled via the TGridColumn ControlType property.

TGridControl.InsertRow Method

```
procedure InsertRow
```

Available In: Visual Client Applications

Use this method to insert a row at the existing RowIndex position in any existing rows in the grid control.

TGridControl.LastColumn Method

```
function LastColumn: Boolean
```

Available In: Visual Client Applications

Use this method to change the current grid column to the last visible column in the grid, if one exists, and update the ColumnIndex property accordingly.

TGridControl.LastRow Method

```
procedure LastRow(ShiftKey, CtrlKey: Boolean=False)
```

Available In: Visual Client Applications

Use this method to change the current grid row to the last row in the grid, if one exists, and update the RowIndex property accordingly.

> **Note**
> The ShiftKey and CtrlKey parameters control how multiple row selection occurs in relation to the row navigation. When the ShiftKey parameter is True, then the range of selected rows is extended to include all rows between the current row index and the row at the new index. When the CtrlKey parameter is True, then the set of selected rows is extended to include the new row at the new index.

TGridControl.LoadRows Method

```
procedure LoadRows(const RowData: String; Append: Boolean=False)
```

Available In: Visual Client Applications

Use this method to load the rows of an un-bound grid from a JSON string. The JSON should be formatted as follows:

```
{   "rows": [
{ "GridColumn1": "Test 1", "GridColumn2": "100", "GridColumn3": "" },
{ "GridColumn1": "Test 2", "GridColumn2": "200", "GridColumn3": "" },
{ "GridColumn1": "Test 3", "GridColumn2": "300", "GridColumn3": "" }
] }
```

TGridControl.MakeColumnVisible Method

```
procedure MakeColumnVisible(AColumn: TGridColumn)
```

Available In: Visual Client Applications

Use this method to ensure that the specified grid column is visible within the grid control.

TGridControl.MakeRowVisible Method

```
procedure MakeRowVisible
```

Available In: Visual Client Applications

Use this method to ensure that the specified grid row is visible within the grid control.

TGridControl.NewColumn Method

```
function NewColumn: TGridColumn
```

Available In: Visual Client Applications

Use this method to create a new column for the grid control and return it as the result. The new grid column will be appended to the existing list of columns defined in the Columns property.

TGridControl.NextColumn Method

```
function NextColumn(AWrap: Boolean=False): Boolean
```

Available In: Visual Client Applications

Use this method to change the current grid column to the next visible column in the grid, if one exists, and update the ColumnIndex property accordingly. The AWrap parameter determines whether to move the current grid row to the next available row, if one exists, if there are no more visible columns in the grid.

TGridControl.NextPage Method

```
procedure NextPage(ShiftKey, CtrlKey: Boolean=False)
```

Available In: Visual Client Applications

Use this method to change the current grid row to the start of the next page of rows in the grid, if one exists, and update the RowIndex property accordingly.

Note
The ShiftKey and CtrlKey parameters control how multiple row selection occurs in relation to the row navigation. When the ShiftKey parameter is True, then the range of selected rows is extended to include all rows between the current row index and the row at the new index. When the CtrlKey parameter is True, then the set of selected rows is extended to include the new row at the new index.

TGridControl.NextRow Method

```
procedure NextRow(ShiftKey, CtrlKey: Boolean=False; CanAppend:
      Boolean=False)
```

Available In: Visual Client Applications

Use this method to change the current grid row to the next row in the grid, if one exists, and update the RowIndex property accordingly. The CanAppend parameter determines whether the grid should automatically call the AppendRow method if the current row index is equal to the last row index for the grid.

Note
The ShiftKey and CtrlKey parameters control how multiple row selection occurs in relation to the row navigation. When the ShiftKey parameter is True, then the range of selected rows is extended to include all rows between the current row index and the row at the new index. When the CtrlKey parameter is True, then the set of selected rows is extended to include the new row at the new index.

TGridControl.PriorColumn Method

```
function PriorColumn(AWrap: Boolean=False): Boolean
```

Available In: Visual Client Applications

Use this method to change the current grid column to the prior visible column in the grid, if one exists, and update the ColumnIndex property accordingly. The AWrap parameter determines whether to move the current grid row to the prior available row, if one exists, if there are no more visible columns in the grid.

TGridControl.PriorPage Method

```
procedure PriorPage(ShiftKey, CtrlKey: Boolean=False)
```

Available In: Visual Client Applications

Use this method to change the current grid row to the end of the next page of rows in the grid, if one exists, and update the RowIndex property accordingly.

Note
The ShiftKey and CtrlKey parameters control how multiple row selection occurs in relation to the row navigation. When the ShiftKey parameter is True, then the range of selected rows is extended to include all rows between the current row index and the row at the new index. When the CtrlKey parameter is True, then the set of selected rows is extended to include the new row at the new index.

TGridControl.PriorRow Method

```
procedure PriorRow(ShiftKey, CtrlKey: Boolean=False)
```

Available In: Visual Client Applications

Use this method to change the current grid row to the prior row in the grid, if one exists, and update the RowIndex property accordingly.

Note
The ShiftKey and CtrlKey parameters control how multiple row selection occurs in relation to the row navigation. When the ShiftKey parameter is True, then the range of selected rows is extended to include all rows between the current row index and the row at the new index. When the CtrlKey parameter is True, then the set of selected rows is extended to include the new row at the new index.

TGridControl.RemoveColumn Method

```
procedure RemoveColumn(AColumn: TGridColumn)
```

Available In: Visual Client Applications

Use this method to remove the specified column from the grid control.

TGridControl.ScrollNext Method

```
procedure ScrollNext
```

Available In: Visual Client Applications

Use this method to scroll the grid down by the height of one row.

TGridControl.ScrollNextPage Method

```
procedure ScrollNextPage
```

Available In: Visual Client Applications

Use this method to scroll the grid down by the height of all visible rows in the grid.

TGridControl.ScrollPrior Method

```
procedure ScrollPrior
```

Available In: Visual Client Applications

Use this method to scroll the grid up by the height of one row.

TGridControl.ScrollPriorPage Method

```
procedure ScrollPriorPage
```

Available In: Visual Client Applications

Use this method to scroll the grid up by the height of all visible rows in the grid.

TGridControl.ScrollTo Method

```
procedure ScrollTo(ARowIndex: Integer)
```

Available In: Visual Client Applications

Use this method to scroll the grid to the specified row in the grid control.

TGridControl.SelectAll Method

```
procedure SelectAll
```

Available In: Visual Client Applications

Use this method to select all of the rows in the grid control.

TGridControl.SelectRange Method

```
procedure SelectRange(AFromRowIndex, AToRowIndex: Integer;
      AClear: Boolean=False)
```

Available In: Visual Client Applications

Use this method to select the specified range of the rows in the grid control. The AClear parameter determines whether the existing set of selected rows should be cleared before the new range of rows is selected.

TGridControl.SetToColumn Method

```
procedure SetToColumn(AIndex: Integer)
```

Available In: Visual Client Applications

Use this method to change the current grid column to the visible column at the specified index in the grid, if one exists, and update the ColumnIndex property accordingly.

TGridControl.SetToRow Method

```
procedure SetToRow(AIndex: Integer; ShiftKey, CtrlKey:
      Boolean=False)
```

Available In: Visual Client Applications

Use this method to change the current grid row to the row at the specified index in the grid, if one exists, and update the RowIndex property accordingly.

Note
The ShiftKey and CtrlKey parameters control how multiple row selection occurs in relation to the row navigation. When the ShiftKey parameter is True, then the range of selected rows is extended to include all rows between the current row index and the row at the new index. When the CtrlKey parameter is True, then the set of selected rows is extended to include the new row at the new index.

TGridControl.ShowColumnControl Method

```
procedure ShowColumnControl
```

Available In: Visual Client Applications

Use this method to show any column controls for the columns in the grid. The control for a grid column is controlled via the TGridColumn ControlType property. This method will change the ColumnControlVisible property to True.

Note
Even if the ColumnControlVisible property is True, it is possible that there won't be any active column controls because row selection is turned on for the grid control or the column doesn't use a control for display or editing. You can use the ColumnControlActive property to determine if any grid column controls are actually active.

TGridControl.SortRows Method

```
procedure SortRows
```

Available In: Visual Client Applications

Use this method to sort an un-bound grid when the Sorted property is True and sort columns have been specified for the grid. The TGridColumn SortDirection property setting controls which columns are sorted, and how.

If no columns have a SortDirection property other than sdNone, then this method does nothing.

The SortCaseInsensitive and SortLocaleInsensitive properties control how the sort is performed.

> **Note**
> The SortRows method only needs to be called once after the sort directions are initially assigned for the grid columns. After that point, any row operations will automatically maintain the active sort.

TGridControl.ToggleSelected Method

```
procedure ToggleSelected(ARowIndex: Integer)
```

Available In: Visual Client Applications

Use this method to toggle the selection state of the row at the specified index.

TGridControl.UpdateCells Method

```
procedure UpdateCells
```

Available In: Visual Client Applications

Use this method to force an update of all grid cells. This is useful for situation where some external state has changed and you want to update one or more grid cells using an event handler for the TGridColumnOnCellUpdate event in one or more grid columns.

10.97 TGridHeader Component

Unit: WebGrids

Inherits From TControl

Available In: Visual Client Applications

The TGridHeader component represents a grid column header. Grid column headers are only visible when the TGrid ColumnHeaders property is True.

Properties	Methods	Events
Alignment		
AllowClick		
Caption		
Font		
Hint		
Wrap		

TGridHeader.Alignment Property

```
property Alignment: TContentAlignment
```

Available In: Visual Client Applications

Specifies the alignment of the caption for the grid column header.

TGridHeader.AllowClick Property

```
property AllowClick: Boolean
```

Available In: Visual Client Applications

Specifies whether the grid column header is clickable. If the header is clickable, then each click will call the ToggleSortDirection method to change the sort direction to the next valid state.

TGridHeader.Caption Property

```
property Caption: String
```

Available In: Visual Client Applications

Specifies the caption for the grid column header.

TGridHeader.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TGridHeader.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TGridHeader.Wrap Property

```
property Wrap: Boolean
```

Available In: Visual Client Applications

Specifies whether the caption should wrap if it exceeds the width of the column header. The default value is False.

10.98 TGridRow Component

Unit: WebGrids

Inherits From TObject

Available In: Visual Client Applications

The TGridRow component represents a grid row in an un-bound grid control, and is accessible via the TGridControl Rows property.

Properties	Methods	Events
Count	Create	
ID	GetJSON	
Value		

TGridRow.Count Property

```
property Count: Integer
```

Available In: Visual Client Applications

Indicates the number of column values in the grid row.

TGridRow.ID Property

```
property ID: Integer
```

Available In: Visual Client Applications

Specifies an ID that unique identifies the grid row.

TGridRow.Value Property

```
property Value[AIndex: Integer]: String
```

Available In: Visual Client Applications

Accesses a specific column value by its column index in the grid row.

TGridRow.Create Method

```
constructor Create(ARows: TGridRows; AID: Integer)
```

Available In: Visual Client Applications

Use this method to create a new instance of the TGridRow class. The ARows parameter indicates the parent grid rows instance that will manage the row, and the AID parameter indicates the unique ID used to identify the grid row.

TGridRow.GetJSON Method

```
function GetJSON: String
```

Available In: Visual Client Applications

Use this method to retrieve the column values of the row as a JSON string. The JSON is formatted as follows:

```
{ "GridColumn1": "Test 1", "GridColumn2": "100", "GridColumn3": "" }
```

10.99 TGridRows Component

Unit: WebGrids

Inherits From TObject

Available In: Visual Client Applications

The TGridRows component represents the grid rows in an un-bound grid control, and are accessible via the TGridControl Rows property.

Properties	Methods	Events
Count	BeginUpdate	
Row	Create	
	EndUpdate	

TGridRows.Count Property

```
property Count: Integer
```

Available In: Visual Client Applications

Indicates the number of rows in an un-bound grid control.

TGridRows.Row Property

```
property Row[AIndex: Integer]: TGridRow
```

Available In: Visual Client Applications

Accesses a grid row by by the specified index.

TGridRows.BeginUpdate Method

```
procedure BeginUpdate
```

Available In: Visual Client Applications

Use this method to begin a batch update to the un-bound grid rows. Batch updates are useful in situations where many changes need to be made to the rows, and triggering grid updates on every change would result in performance issues. This method is reference-counted and every time it is called, an internal counter is incremented. Every time the EndUpdate method is called, the counter is decremented. Once the counter reaches zero, the grid will be notified that it needs to update its contents.

TGridRows.Create Method

```
constructor Create(AGrid: TGridControl)
```

Available In: Visual Client Applications

Use this method to create a new instance of the TGridRows class. The AGrid parameter indicates the parent grid control instance that will manage the rows.

TGridRows.EndUpdate Method

```
procedure EndUpdate
```

Available In: Visual Client Applications

Use this method to end a batch update to the un-bound grid rows. Batch updates are useful in situations where many changes need to be made to the rows, and triggering grid updates on every change would result in performance issues. This method is reference-counted and every time it is called, an internal counter is decremented. Every time the BeginUpdate method is called, the counter is incremented. Once the counter reaches zero, the grid will be notified that it needs to update its contents.

10.100 TGroupPanel Component

Unit: WebCtnrs

Inherits From TGroupPanelControl

Available In: Visual Client Applications

The TGroupPanel component represents a group panel control for organizing sets of controls like radio buttons into a container with a caption.

Properties	Methods	Events
ActivateOnClick		OnAnimationComplete
Background		OnAnimationsComplete
Caption		OnCaptureEnd
Cursor		OnCaptureStart
Font		OnCapturing
Hint		OnClick
InsetShadow		OnContextMenu
Opacity		OnDblClick
OutsetShadow		OnHide
Padding		OnKeyDown
TabOrder		OnKeyPress
TabStop		OnKeyUp
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TGroupPanel.ActivateOnClick Property

```
property ActivateOnClick: Boolean
```

Available In: Visual Client Applications

Specifies whether the control should automatically be brought to the front when it, or any child controls, are clicked.

TGroupPanel.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

TGroupPanel.Caption Property

```
property Caption: String
```

Available In: Visual Client Applications

Specifies the caption for the group panel control.

TGroupPanel.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TGroupPanel.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TGroupPanel.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TGroupPanel.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Available In: Visual Client Applications

Specifies the inset shadow for the control.

TGroupPanel.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TGroupPanel.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Available In: Visual Client Applications

Specifies the outset shadow for the control.

TGroupPanel.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

TGroupPanel.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TGroupPanel.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TGroupPanel.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TGroupPanel.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TGroupPanel.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TGroupPanel.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TGroupPanel.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TGroupPanel.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TGroupPanel.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TGroupPanel.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TGroupPanel.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TGroupPanel.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and the user presses a key or key combination.

TGroupPanel.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and presses/releases a key or key combination.

TGroupPanel.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and the user releases a key or key combination.

TGroupPanel.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TGroupPanel.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TGroupPanel.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TGroupPanel.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TGroupPanel.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TGroupPanel.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TGroupPanel.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TGroupPanel.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TGroupPanel.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TGroupPanel.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TGroupPanel.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TGroupPanel.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.101 TGroupPanelControl Component

Unit: WebCtnrs

Inherits From TControl

Available In: Visual Client Applications

The TGroupPanelControl control is the base class for group panels, and contains all of the core group panel functionality in the form of public methods and protected properties/events that descendant classes can use to create customized group panels.

Properties	Methods	Events

10.102 THeaderPanel Component

Unit: WebCtnrs

Inherits From THeaderPanelControl

Available In: Visual Client Applications

The THeaderPanel component represents a header panel control that acts as a container for other controls.

Properties	Methods	Events
Background		OnAnimationComplete
Border		OnAnimationsComplete
Corners		OnCaptureEnd
Cursor		OnCaptureStart
Hint		OnCapturing
InsetShadow		OnClick
Opacity		OnContextMenu
OutsetShadow		OnDblClick
Padding		OnHide
TabOrder		OnKeyDown
TabStop		OnKeyPress
		OnKeyUp
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

THeaderPanel.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

THeaderPanel.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

THeaderPanel.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

THeaderPanel.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

THeaderPanel.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

THeaderPanel.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Available In: Visual Client Applications

Specifies the inset shadow for the control.

THeaderPanel.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

THeaderPanel.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Available In: Visual Client Applications

Specifies the outset shadow for the control.

THeaderPanel.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

THeaderPanel.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

THeaderPanel.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

THeaderPanel.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

THeaderPanel.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

THeaderPanel.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

THeaderPanel.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

THeaderPanel.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

THeaderPanel.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

THeaderPanel.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

THeaderPanel.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

THeaderPanel.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

THeaderPanel.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and the user presses a key or key combination.

THeaderPanel.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and presses/releases a key or key combination.

THeaderPanel.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and the user releases a key or key combination.

THeaderPanel.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

THeaderPanel.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

THeaderPanel.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

THeaderPanel.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

THeaderPanel.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

THeaderPanel.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

THeaderPanel.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

THeaderPanel.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

THeaderPanel.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

THeaderPanel.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

THeaderPanel.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

THeaderPanel.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.103 THeaderPanelControl Component

Unit: WebCtnrs

Inherits From TControl

Available In: Visual Client Applications

The THeaderPanelControl control is the base class for header panels, and contains all of the core header panel functionality in the form of public methods and protected properties/events that descendant classes can use to create customized header panels.

Properties	Methods	Events

10.104 THeaders Component

Unit: WebCore

Inherits From TObject

Available In: Client and Server Applications

The THeaders class is used by the TMIMEContentPart and TMIMEContentclasses in the TMailer class for representing the MIME headers for of multi-part MIME content being sent with the email.

Properties	Methods	Events
Count	Assign	
Names	Clear	
Text	Create	
ValueFromIndex		
Values		

THeaders.Count Property

```
property Count: Integer
```

Available In: Client and Server Applications

Indicates the total number of headers.

THeaders.Names Property

```
property Names[Index: Integer]: String
```

Available In: Client and Server Applications

Indicates the name of the header at the specified index.

THeaders.Text Property

```
property Text: String
```

Available In: Client and Server Applications

Gets or sets the headers as a set of name-value pairs (<Name>: <Value>) with CRLF delimiters between each name-value pair.

THeaders.ValueFromIndex Property

```
property ValueFromIndex[Index: Integer]: String
```

Available In: Client and Server Applications

Gets or sets the header value at the specified index.

THeaders.Values Property

```
property Values[const Name: String]: String
```

Available In: Client and Server Applications

Gets or sets the header value with the specified name.

THeaders.Assign Method

```
procedure Assign(Headers: THeaders)

procedure Assign(Strings: TStrings)
```

Available In: Client and Server Applications

Use this method to assign the specified headers. All existing headers will be deleted and overwritten with the source headers.

THeaders.Clear Method

```
procedure Clear
```

Available In: Client and Server Applications

Use this method to delete all headers.

THeaders.Create Method

```
constructor Create
```

Available In: Client and Server Applications

Use this method to create a new instance of the THeaders class.

> **Note**
> Because the headers are automatically created and included as part of a MIME part, you will most likely never need to call this method.

10.105 THiddenInputElement Component

Unit: WebUI

Inherits From TInputElement

Available In: Visual Client Applications

The THiddenInputElement class is the element class for hidden HTML form input elements. Hidden form input elements are useful for programmatically adding HTML form values, and the client component library uses them for custom controls like the virtual TListBox control that have no counterpart in standard HTML form input elements.

> **Note**
> This element does not provide support for hidden HTML form inputs at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events

10.106 THTMLForm Component

Unit: WebBrwsr

Inherits From THTMLFormControl

Available In: Visual Client Applications

The THTMLForm component represents an HTML form control. HTML forms are containers for any input controls whose input values can be submitted as HTML form values to the web server.

Properties	Methods	Events
Background		OnAnimationComplete
Border		OnAnimationsComplete
Corners		OnCaptureEnd
Cursor		OnCaptureStart
Encoding		OnCapturing
FormValues		OnClick
InsetShadow		OnContextMenu
Method		OnDblClick
Opacity		OnHide
Output		OnMouseDown
Padding		OnMouseEnter
TabOrder		OnMouseLeave
TabStop		OnMouseMove
URL		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnSubmit
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

THTMLForm.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background for the control.

THTMLForm.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

THTMLForm.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

THTMLForm.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

THTMLForm.Encoding Property

```
property Encoding: THTMLFormEncoding
```

Available In: Visual Client Applications

Specifies the type of MIME content encoding for the form data that is sent when the Submit method is called.

THTMLForm.FormValues Property

```
property FormValues: TStrings
```

Available In: Visual Client Applications

Specifies any additional form values to be submitted along with values from the input controls. The form values should be specified as key-value pairs.

> **Note**
> These additional form values are handled as hidden input controls within the HTML form.

THTMLForm.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Available In: Visual Client Applications

Specifies the inset shadow for the control.

THTMLForm.Method Property

```
property Method: THTMLFormMethod
```

Available In: Visual Client Applications

Specifies the HTTP method used for the form submittal when the Submit method is called.

THTMLForm.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

THTMLForm.Output Property

```
property Output: TBrowser
```

Available In: Visual Client Applications

Specifies a TBrowser instance that should accept all output from the form submittal request to the web server when the Submit method is called. The default value is nil.

THTMLForm.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

THTMLForm.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

THTMLForm.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

THTMLForm.URL Property

```
property URL: String
```

Available In: Visual Client Applications

Specifies the URL to use for the form submittal when the Submit method is called. The default value is ''.

THTMLForm.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

THTMLForm.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

THTMLForm.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

THTMLForm.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

THTMLForm.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

THTMLForm.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

THTMLForm.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

THTMLForm.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

THTMLForm.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

THTMLForm.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

THTMLForm.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

THTMLForm.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

THTMLForm.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

THTMLForm.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

THTMLForm.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

THTMLForm.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

THTMLForm.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

THTMLForm.OnSubmit Event

```
property OnSubmit: TNotifyEvent
```

Available In: Visual Client Applications

This event is fired when the HTML form is submitted.

THTMLForm.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

THTMLForm.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

THTMLForm.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

THTMLForm.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.107 THTMLFormControl Component

Unit: WebBrwsr

Inherits From TControl

Available In: Visual Client Applications

The THTMLFormControl control is the base class for HTML forms, and contains all of the core HTML form functionality in the form of public methods and protected properties/events that descendant classes can use to create customized HTML forms.

Properties	Methods	Events
	Reset	
	Submit	

THTMLFormControl.Reset Method

```
procedure Reset
```

Available In: Visual Client Applications

Use this method to reset the HTML form values associated with any input controls contained within the HTML form control. Resetting the form values will set them all to an empty string ('').

THTMLFormControl.Submit Method

```
procedure Submit
```

Available In: Visual Client Applications

Use this method to submit the HTML form values associated with any input controls contained within the HTML form control.

10.108 THTMLLabel Component

Unit: WebLabels

Inherits From THTMLLabelControl

Available In: Visual Client Applications

The THTMLLabel component represents an HTML label control. An HTML label control displays arbitrary HTML content using a specific font and formatting, including alignment and wrapping.

Properties	Methods	Events
AutoSize		OnAnimationComplete
Background		OnAnimationsComplete
Border		OnCaptureEnd
Content		OnCaptureStart
Corners		OnCapturing
Cursor		OnClick
DataColumn		OnContextMenu
DataSet		OnDblClick
Font		OnHide
Format		OnMouseDown
Hint		OnMouseEnter
Opacity		OnMouseLeave
OutsetShadow		OnMouseMove
Padding		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

THTMLLabel.AutoSize Property

```
property AutoSize: TAutoSize
```

Available In: Visual Client Applications

Specifies how (if at all) the control should automatically be sized based upon the Content and Format properties.

THTMLLabel.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

THTMLLabel.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

THTMLLabel.Content Property

```
property Content: TContent
```

Available In: Visual Client Applications

Specifies the HTML content to display in the control. The default value is ''.

THTMLLabel.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

THTMLLabel.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

THTMLLabel.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

THTMLLabel.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

THTMLLabel.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

THTMLLabel.Format Property

```
property Format: TFormat
```

Available In: Visual Client Applications

Specifies the content formatting to use for the control's Content.

THTMLLabel.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

THTMLLabel.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

THTMLLabel.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Available In: Visual Client Applications

Specifies the outset shadow for the control.

THTMLLabel.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

THTMLLabel.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

THTMLLabel.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

THTMLLabel.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

THTMLLabel.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

THTMLLabel.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

THTMLLabel.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

THTMLLabel.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

THTMLLabel.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

THTMLLabel.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

THTMLLabel.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

THTMLLabel.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

THTMLLabel.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

THTMLLabel.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

THTMLLabel.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

THTMLLabel.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

THTMLLabel.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

THTMLLabel.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

THTMLLabel.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

THTMLLabel.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

THTMLLabel.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

THTMLLabel.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.109 THTMLLabelControl Component

Unit: WebLabels

Inherits From TBindableColumnControl

Available In: Visual Client Applications

The THTMLLabelControl control is the base class for HTML label controls, and contains all of the core HTML label functionality in the form of public methods and protected properties/events that descendant classes can use to create customized HTML label controls.

Properties	Methods	Events

10.110 THTTPContentPart Component

Unit: WebSrvr

Inherits From TExternalObject

Available In: Server Applications

The THTTPContentPart class is used by the THTTPContentParts class to represent one part of the multi-part content in a web server request.

Properties	Methods	Events
Content	Create	
ContentCharSet		
ContentDisposition		
ContentFileName		
ContentName		
ContentStream		
ContentTransferEncoding		
ContentType		
Headers		

THTTPContentPart.Content Property

```
property Content: String
```

Available In: Server Applications

Contains any textual content included with the content part. If the ContentType property is set to one of the following:

MIME Type	Description
text/plain	Plain text
text/html	HTML
text/xml	XML
application/xml	XML
application/json	JSON
application/javascript	JavaScript
application/pdf	PDF document

then the part content will be available in this property. If the ContentType property contains any other value, then the part content will need to be accessed using the ContentStream property.

Note
If the character set is set as part of the ContentType property using the **charset** attribute, then it must be set to "utf-8". If the character set is set to a different value, then the content part content will need to be accessed using the ContentStream property.

THTTPContentPart.ContentCharSet Property

```
property ContentCharSet: String
```

Available In: Server Applications

Indicates the **charset** attribute included with the **Content-Type** content part header.

THTTPContentPart.ContentDisposition Property

```
property ContentDisposition: String
```

Available In: Server Applications

Indicates the value of the **Content-Disposition** content part header.

THTTPContentPart.ContentFileName Property

```
property ContentFileName: String
```

Available In: Server Applications

Indicates the **filename** attribute included with the **Content-Disposition** content part header.

THTTPContentPart.ContentName Property

```
property ContentName: String
```

Available In: Server Applications

Indicates the **name** attribute included with the **Content-Disposition** content part header.

THTTPContentPart.ContentStream Property

```
property ContentStream: TStream
```

Available In: Server Applications

Makes any non-textual content included with the content part accessible via a TStream instance. Please see the Content property for more information on what constitutes textual content in the content part.

THTTPContentPart.ContentTransferEncoding Property

```
property ContentTransferEncoding: String
```

Available In: Server Applications

Indicates the value of the **Content-Transfer-Encoding** content part header.

THTTPContentPart.ContentType Property

```
property ContentType: String
```

Available In: Server Applications

Indicates the value of the **Content-Type** content part header.

THTTPContentPart.Headers Property

```
property Headers: THTTPContentPartHeaders
```

Available In: Server Applications

Accesses all headers included with the content part.

THTTPContentPart.Create Method

```
constructor Create(Owner: THTTPContentParts)
```

Available In: Server Applications

Use this method to create a new THTTPContentPart instance.

> **Note**
> Because the headers are automatically created and included as part of the multi-part content for the incoming web server request, you will most likely never need to call this method.

10.111 THTTPContentPartHeaders Component

Unit: WebSrvr

Inherits From TExternalObject

Available In: Server Applications

The THTTPContentPartHeaders class is used by the THTTPContentPart class for representing the headers for one part of the multi-part content in a web server request.

Properties	Methods	Events
Count	Clear	
Names	Create	
Text		
ValueFromIndex		
Values		

THTTPContentPartHeaders.Count Property

```
property Count: Integer
```

Available In: Server Applications

Indicates the total number of content part headers.

THTTPContentPartHeaders.Names Property

```
property Names[Index: Integer]: String
```

Available In: Server Applications

Indicates the name of the content part header at the specified index.

THTTPContentPartHeaders.Text Property

```
property Text: String
```

Available In: Server Applications

Gets or sets the content part headers as a set of name-value pairs (<Name>: <Value>) with CRLF delimiters between each name-value pair.

THTTPContentPartHeaders.ValueFromIndex Property

```
property ValueFromIndex[Index: Integer]: String
```

Available In: Server Applications

Gets or sets the content part header value at the specified index.

THTTPContentPartHeaders.Values Property

```
property Values[const Name: String]: String
```

Available In: Server Applications

Gets or sets the content part header value with the specified name.

THTTPContentPartHeaders.Clear Method

```
procedure Clear
```

Available In: Server Applications

Use this method to delete all content part headers.

THTTPContentPartHeaders.Create Method

```
constructor Create
```

Available In: Server Applications

Use this method to create a new instance of the THTTPContentPartHeaders class.

> **Note**
> Because the headers are automatically created and included as part of the multi-part content for the incoming web server request, you will most likely never need to call this method.

10.112 THTTPContentParts Component

Unit: WebSrvr

Inherits From TExternalObject

Available In: Server Applications

The THTTPContentParts class is used by the TWebServerRequest class to represent the multi-part content for an incoming web server request.

Properties	Methods	Events
Count	Add	
Items	Clear	
	Create	
	Delete	

THTTPContentParts.Count Property

```
property Count: Integer
```

Available In: Server Applications

Indicates the total number of content parts.

THTTPContentParts.Items Property

```
property Items[AIndex: Integer]: THTTPContentPart
```

Available In: Server Applications

Accesses the content part at the specified index.

THTTPContentParts.Add Method

```
function Add: THTTPContentPart
```

Available In: Server Applications

Use this method to add a new content part.

THTTPContentParts.Clear Method

```
procedure Clear
```

Available In: Server Applications

Use this method to delete all content parts.

THTTPContentParts.Create Method

```
constructor Create
```

Available In: Server Applications

Use this method to create a new instance of the THTTPContentParts class.

> **Note**
> Because the content parts are automatically created and included as part of the incoming web server request, you will most likely never need to call this method.

THTTPContentParts.Delete Method

```
procedure Delete(Index: Integer)
```

Available In: Server Applications

Use this method to delete the content part at the specified index.

10.113 THTTPCookie Component

Unit: WebSrvr

Inherits From TExternalObject

Available In: Server Applications

The THTTPCookie class is used with the THTTPCookies class to represent an HTTP cookie in the set of HTTP cookies included with an incoming web server request, as well as the HTTP cookies that are set in response to the web server request.

HTTP cookies are text items set by server applications and/or native server modules and returned to the browser, where they are cached according to the THTTPCookie Expires and MaxAge properties. In addition, the cookies are sent back to the same server applications and/or native server modules with any subsequent web server requests. Which cookies are sent with which server requests is controlled by the THTTPCookie Domain, Path, and Secure properties.

Properties	Methods	Events
CreatedOn	Create	
Domain		
Expires		
HTTPOnly		
MaxAge		
Name		
Path		
Secure		
Text		
Value		

THTTPCookie.CreatedOn Property

```
property CreatedOn: DateTime
```

Available In: Server Applications

Indicates the date/time when the cookie was originally created.

THTTPCookie.Domain Property

```
property Domain: String
```

Available In: Server Applications

Specifies the domain that the cookie should be used with. If the cookie domain does not match the domain of the current web server request, then it will not be sent by the browser to the web server as part of the request. The default value is blank ('').

THTTPCookie.Expires Property

```
property Expires: DateTime
```

Available In: Server Applications

Specifies the date/time when the cookie is set to expire. The default value is blank (0), which indicates that the cookie is a per-session cookie.

This property is the alternate of the MaxAge property. Whereas this property allows you to set the actual expiration date/time for the cookie, the MaxAge property allows you to specify how long you want the cookie to be available before it expires.

THTTPCookie.HTTPOnly Property

```
property HTTPOnly: Boolean
```

Available In: Server Applications

Specifies that the cookie should only be visible to the browser and not visible to any JavaScript code executing in the browser. The default value is False.

THTTPCookie.MaxAge Property

```
property MaxAge: Integer
```

Available In: Server Applications

Specifies how long, in seconds, the cookie should be available before it expires. The default value is 0, which indicates that the cookie is a per-session cookie.

This property is the alternate of the Expires property. Whereas this property allows you to set how long you want the cookie to be available before it expires, the Expires property allows you to set the the actual expiration date/time for the cookie.

THTTPCookie.Name Property

```
property Name: String
```

Available In: Server Applications

Indicates the name of the cookie.

THTTPCookie.Path Property

```
property Path: String
```

Available In: Server Applications

Specifies the path that the cookie should be used with. If the cookie path is not contained within the path of the current web server request, then it will not be sent by the browser to the web server as part of the request. The default value is blank ('').

THTTPCookie.Secure Property

```
property Secure: Boolean
```

Available In: Server Applications

Specifies that the cookie should only be used with secure HTTPS requests. If the current web server request was not a secure HTTPS request, then the cookie will not be sent by the browser to the web server as part of the request. The default value is blank ('').

THTTPCookie.Text Property

```
property Text: String
```

Available In: Server Applications

Gets or sets the cookie using its raw HTTP **Set-Cookie** response header format.

THTTPCookie.Value Property

```
property Value: String
```

Available In: Server Applications

Specifies the cookie value. The default value is blank ('').

THTTPCookie.Create Method

```
constructor Create(const AName: String)
```

Available In: Server Applications

Use this method to create a new THTTPCookie instance with the specified name.

10.114 THTTPCookies Component

Unit: WebSrvr

Inherits From TExternalObject

Available In: Server Applications

The THTTPCookies class is by the TWebServerRequest class to represent the set of HTTP cookies included with an incoming web server request, as well as the HTTP cookies that are set in response to the web server request. When a request is sent to the web server, any included cookies will be available in the TWebServerRequest RequestCookies property.

HTTP cookies are text items set by server applications and/or native server modules and returned to the browser, where they are cached according to the THTTPCookie Expires and MaxAge properties. In addition, the cookies are sent back to the same server applications and/or native server modules with any subsequent web server requests. Which cookies are sent with which server requests is controlled by the THTTPCookie Domain, Path, and Secure properties.

Properties	Methods	Events
Cookie	Add	
Count	Assign	
	Clear	
	Create	
	Exists	
	GetNames	
	Remove	

THTTPCookies.Cookie Property

```
property Cookie[const Name: String]: THTTPCookie
```

Available In: Server Applications

Retrieves the cookie that matches the specified name, or nil if no such cookie exists.

THTTPCookies.Count Property

```
property Count: Integer
```

Available In: Server Applications

Indicates the total number of cookies.

THTTPCookies.Add Method

```
function Add(const Name: String): THTTPCookie
```

Available In: Server Applications

Use this method to add a new cookie with the specified name to the set of cookies.

THTTPCookies.Assign Method

```
procedure Assign(Cookies: THTTPCookies)
```

Use this method to assign the specified cookies. All existing cookies will be deleted and overwritten with the source cookies.

THTTPCookies.Clear Method

```
procedure Clear
```

Available In: Server Applications

Use this method to delete all cookies in the set.

THTTPCookies.Create Method

```
constructor Create
```

Available In: Server Applications

Use this method to create a new instance of the THTTPCookies class.

Note
Because the cookies are automatically created and included as part of the incoming web server request, you will most likely never need to call this method.

THTTPCookies.Exists Method

```
function Exists(const Name: String): Boolean
```

Available In: Server Applications

Use this method to determine if a cookie with the specified name exists in the set of cookies.

THTTPCookies.GetNames Method

```
function GetNames: array of String
```

Available In: Server Applications

Use this method to get an array of strings that contains the names of all of the cookies in the set.

THTTPCookies.Remove Method

```
procedure Remove(const Name: String)
```

Available In: Server Applications

Use this method to remove the cookie with the specified name from the set of cookies.

10.115 THTTPFormValues Component

Unit: WebSrvr

Inherits From TExternalObject

Available In: Server Applications

The THTTPFormValues class is used by the TWebServerRequest class for representing any HTML form values included with an incoming web server request. When an HTML form is submitted to the web server, any form values will be available in the TWebServerRequest RequestFormValues property.

> **Note**
> Any file uploads included as part of a multi-part HTML form submittal request will be available in the RequestContentParts property.

Please see the Using HTML Forms topic for more information on how to submit HTML forms to a server application or native server module.

Properties	Methods	Events
Count	Clear	
Names	Create	
Text		
ValueFromIndex		
Values		

THTTPFormValues.Count Property

```
property Count: Integer
```

Available In: Server Applications

Indicates the total number of HTML form values.

THTTPFormValues.Names Property

```
property Names[Index: Integer]: String
```

Available In: Server Applications

Indicates the name of the HTML form value at the specified index.

THTTPFormValues.Text Property

```
property Text: String
```

Available In: Server Applications

Gets or sets the HTML form values as a set of name-value pairs (<Name>=<Value>) with CRLF delimiters between each name-value pair.

THTTPFormValues.ValueFromIndex Property

```
property ValueFromIndex[Index: Integer]: String
```

Available In: Server Applications

Gets or sets the HTML form value at the specified index.

THTTPFormValues.Values Property

```
property Values[const Name: String]: String
```

Available In: Server Applications

Gets or sets the HTML form value with the specified name.

THTTPFormValues.Clear Method

```
procedure Clear
```

Available In: Server Applications

Use this method to delete all HTML form values.

THTTPFormValues.Create Method

```
constructor Create
```

Available In: Server Applications

Use this method to create a new instance of the THTTPFormValues class.

> **Note**
> Because the HTML form values are automatically created and included as part of the incoming web server request, you will most likely never need to call this method.

10.116 THTTPHeaders Component

Unit: WebSrvr

Inherits From TExternalObject

Available In: Server Applications

The THTTPHeaders class is used by the TWebServerRequest class for representing the request/response headers for an incoming web server request. When a request is sent to the web server, any HTTP request headers will be available in the TWebServerRequest RequestHeaders property, and the server application can set any HTTP response headers using the TWebServerRequest ResponseHeaders property.

Properties	Methods	Events
Count	Clear	
Names	Create	
Text		
ValueFromIndex		
Values		

THTTPHeaders.Count Property

```
property Count: Integer
```

Available In: Server Applications

Indicates the total number of headers.

THTTPHeaders.Names Property

```
property Names[Index: Integer]: String
```

Available In: Server Applications

Indicates the name of the header at the specified index.

THTTPHeaders.Text Property

```
property Text: String
```

Available In: Server Applications

Gets or sets the headers as a set of name-value pairs (<Name>: <Value>) with CRLF delimiters between each name-value pair.

THTTPHeaders.ValueFromIndex Property

```
property ValueFromIndex[Index: Integer]: String
```

Available In: Server Applications

Gets or sets the header value at the specified index.

THTTPHeaders.Values Property

```
property Values[const Name: String]: String
```

Available In: Server Applications

Gets or sets the header value with the specified name.

THTTPHeaders.Clear Method

```
procedure Clear
```

Available In: Server Applications

Use this method to delete all headers.

THTTPHeaders.Create Method

```
constructor Create
```

Available In: Server Applications

Use this method to create a new instance of the THTTPHeaders class.

> **Note**
> Because the headers are automatically created and included as part of the incoming web server request, you will most likely never need to call this method.

10.117 THTTPParameters Component

Unit: WebSrvr

Inherits From TExternalObject

Available In: Server Applications

The THTTPParameters class is used by the TWebServerRequest class for representing the URL parameters for an incoming web server request. When a request is sent to the web server, any URL parameters in the HTTP request will be available in the TWebServerRequest RequestParameters property.

Properties	Methods	Events
Count	Assign	
Names	Clear	
Text	Create	
ValueFromIndex		
Values		

THTTPParameters.Count Property

```
property Count: Integer
```

Available In: Server Applications

Indicates the total number of parameters.

THTTPParameters.Names Property

```
property Names[Index: Integer]: String
```

Available In: Server Applications

Indicates the name of the parameter at the specified index.

THTTPParameters.Text Property

```
property Text: String
```

Available In: Server Applications

Gets or sets the parameters as a set of name-value pairs (<Name>=<Value>) in URL format:

```
?<Name>=<Value>[&<Name>=<Value>[&<Name>=<Value>]]
```

Note
The value returned by this property is not encoded and usable as part of a valid URL. Please see the THTTPParameters EncodedText property for the encoded version of this property.

THTTPParameters.ValueFromIndex Property

```
property ValueFromIndex[Index: Integer]: String
```

Available In: Server Applications

Gets or sets the parameter value at the specified index.

THTTPParameters.Values Property

```
property Values[const Name: String]: String
```

Available In: Server Applications

Gets or sets the parameter value with the specified name.

THTTPParameters.Assign Method

```
procedure Assign(Parameters: THTTPParameters)
```

Available In: Server Applications

Use this method to assign the specified parameters. All existing parameters will be deleted and overwritten with the source parameters.

THTTPParameters.Clear Method

```
procedure Clear
```

Available In: Server Applications

Use this method to delete all parameters.

THTTPParameters.Create Method

```
constructor Create
```

Available In: Server Applications

Use this method to create a new instance of the THTTPParameters class.

> **Note**
> Because the parameters are automatically created and included as part of the incoming web server request, you will most likely never need to call this method.

10.118 THTTPPathComponents Component

Unit: WebSrvr

Inherits From TExternalObject

Available In: Server Applications

The THTTPPathComponents class is used by the TWebServerRequest class for representing the URL path components for an incoming web server request. When a request is sent to the web server, any URL path components in the HTTP request will be available in the TWebServerRequest RequestPathComponents property.

Properties	Methods	Events
Components	Assign	
Count	Clear	
Delimiter	Create	
Text		

THTTPPathComponents.Components Property

```
property Components[Index: Integer]: String
```

Available In: Server Applications

Gets the path component at the specified index.

THTTPPathComponents.Count Property

```
property Count: Integer
```

Available In: Server Applications

Indicates the total number of path components.

THTTPPathComponents.Delimiter Property

```
property Delimiter: String
```

Available In: Server Applications

Specifies the delimiter to use with the Text property. The default value is "/".

THTTPPathComponents.Text Property

```
property Text: String
```

Available In: Server Applications

Gets or sets the path components as a set of values delimited by the Delimiter property.

THTTPPathComponents.Assign Method

```
procedure Assign(PathComponents: THTTPPathComponents)
```

Available In: Server Applications

Use this method to assign all of the specified path components. All path components will be deleted and overwritten with the source path components.

THTTPPathComponents.Clear Method

```
procedure Clear
```

Available In: Server Applications

Use this method to delete all path components.

THTTPPathComponents.Create Method

```
constructor Create
```

Available In: Server Applications

Use this method to create a new instance of the THTTPPathComponents class.

> **Note**
> Because the path components are automatically created and included as part of the incoming web server request, you will most likely never need to call this method.

10.119 THTTPServerRequest Component

Unit: WebSrvr

Inherits From TExternalObject

Available In: Server Applications

The THTTPServerRequest class is the base class for the TWebServerRequest class that is used to represent an incoming web server request for server applications.

Properties	Methods	Events
RequestClientAddress	AcceptsContentEncoding	
RequestContent	SendContent	
RequestContentLength	SendContentFile	
RequestContentParts	SendContentHeader	
RequestContentStream	SendContentStream	
RequestContentType	SendCustomContent	
RequestCookies	SendCustomContentHeader	
RequestFormValues	SendCustomContentStream	
RequestHeaders	SendError	
RequestHost	SendRedirect	
RequestMethod		
RequestMethodName		
RequestParameters		
RequestPathComponents		
RequestSecure		
RequestURL		
RequestURLParams		
RequestURLPath		
ResponseCookies		
ResponseHeaders		
ResponseSessionCookies		

THTTPServerRequest.RequestClientAddress Property

```
property RequestClientAddress: String
```

Available In: Server Applications

Indicates the public IP address of the client making the request to the web server.

THTTPServerRequest.RequestContent Property

```
property RequestContent: String
```

Available In: Server Applications

Contains any textual content included with the web server request. If the RequestContentType property is set to one of the following:

MIME Type	Description
text/plain	Plain text
text/html	HTML
text/xml	XML
application/xml	XML
application/json	JSON
application/javascript	JavaScript
application/pdf	PDF document

then the included content will be available in this property. If the RequestContentType property contains any other value, then the included content will need to be accessed using the RequestContentStream property.

Note
If the character set is set as part of the RequestContentType property using the **charset** attribute, then it must be set to "utf-8". If the character set is set to a different value, then the included content will need to be accessed using the RequestContentStream property.

THTTPServerRequest.RequestContentLength Property

```
property RequestContentLength: Integer
```

Available In: Server Applications

Indicates the size, in bytes, of any content included with the web server request. This value is the integer value specified in the **Content-Length** header in the RequestHeaders property.

THTTPServerRequest.RequestContentParts Property

```
property RequestContentParts: THTTPContentParts
```

Available In: Server Applications

If the web server request includes multi-part content, then this property will contain the various parts that make up the content.

THTTPServerRequest.RequestContentStream Property

```
property RequestContentStream: TStream
```

Available In: Server Applications

Makes any non-textual content included with the web server request accessible via a TStream instance. Please see the RequestContent property for more information on what constitutes textual content in the web server request.

THTTPServerRequest.RequestContentType Property

```
property RequestContentType: String
```

Available In: Server Applications

Indicates the content type of any content included with the web server request. This value is the string value specified in the **Content-Type** header in the RequestHeaders property.

THTTPServerRequest.RequestCookies Property

```
property RequestCookies: THTTPCookies
```

Available In: Server Applications

Contains any HTTP cookies included with the web server request.

THTTPServerRequest.RequestFormValues Property

```
property RequestFormValues: THTTPFormValues
```

Available In: Server Applications

If the web server request is an HTML form submittal, this property will contain the submitted form values.

> **Note**
> Any file uploads included as part of a multi-part HTML form submittal request will be available in the RequestContentParts property.

THTTPServerRequest.RequestHeaders Property

```
property RequestHeaders: THTTPHeaders
```

Available In: Server Applications

Contains any HTTP headers included with the web server request.

THTTPServerRequest.RequestHost Property

```
property RequestHost: String
```

Available In: Server Applications

Indicates the target host for the web server request. This value is the string value specified in the **Host** header in the RequestHeaders property.

THTTPServerRequest.RequestMethod Property

```
property RequestMethod: THTTPMethod
```

Available In: Server Applications

Indicates the HTTP method for the web server request.

THTTPServerRequest.RequestMethodName Property

```
property RequestMethodName: String
```

Available In: Server Applications

Indicates the textual version of the HTTP method for the web server request.

THTTPServerRequest.RequestParameters Property

```
property RequestParameters: THTTPParameters
```

Available In: Server Applications

Contains any URL parameters included with the web server request.

THTTPServerRequest.RequestPathComponents Property

```
property RequestPathComponents: THTTPPathComponents
```

Available In: Server Applications

Contains all path components contained in the RequestURLPath property.

THTTPServerRequest.RequestSecure Property

```
property RequestSecure: Boolean
```

Available In: Server Applications

Indicates whether the web server request as a secure HTTPs or insecure HTTP request.

THTTPServerRequest.RequestURL Property

```
property RequestURL: String
```

Available In: Server Applications

Contains the full URL for the web server request, including any URL parameters.

THTTPServerRequest.RequestURLParams Property

```
property RequestURLParams: String
```

Available In: Server Applications

Contains any URL parameters included as part of the RequestURL property.

THTTPServerRequest.RequestURLPath Property

```
property RequestURLPath: String
```

Available In: Server Applications

Contains the URL path included as part of the RequestURL property.

> **Note**
> This property only contains the path and does not contain the origin (protocol, host, port) or any parameters.

THTTPServerRequest.ResponseCookies Property

```
property ResponseCookies: THTTPCookies
```

Available In: Server Applications

Specifies any cookies to be included with the response to the web server request.

THTTPServerRequest.ResponseHeaders Property

```
property ResponseHeaders: THTTPHeaders
```

Available In: Server Applications

Specifies any headers to be included with the response to the web server request.

THTTPServerRequest.ResponseSessionCookies Property

```
property ResponseSessionCookies: THTTPCookies
```

Available In: Server Applications

Specifies any session-only cookies to be included with the response to the web server request.

Note
Please use this property instead of the ResponseCookies property when returning session-only cookies. The web server request automatically sets the THTTPCookie Expires property to a default value of 1 year from the current date/time for all cookies contained in the ResponseCookies property.

THTTPServerRequest.AcceptsContentEncoding Method

```
function AcceptsContentEncoding(const ContentEncoding: String):
      Boolean
```

Available In: Server Applications

Use this method to determine if the **Accept-Encoding** header included with the web server request contains the specified content encoding. If an **Accept-Encoding** header was not included with the web server request, then this method will always return False.

THTTPServerRequest.SendContent Method

```
procedure SendContent(const Content: String; StatusCode:
      Integer=HTTP_OK; const StatusMessage: String='')
```

Available In: Server Applications

Sends a UTF-8-encoded text response with a **Content-Type** header of "text/html; charset=utf-8" along with an optional status code and message.

THTTPServerRequest.SendContentFile Method

```
procedure SendContentFile(const ContentFileName: String; const
      ContentType: String=''; const ContentEncoding: String=''; const
      ContentDisposition: String='')
```

Available In: Server Applications

Sends a file response with a custom content type, encoding, and disposition.

> **Note**
> Information If the content type is not specified, then the web server will attempt to determine the content type based upon the file extension.

THTTPServerRequest.SendContentHeader Method

```
procedure SendContentHeader(StatusCode: Integer=HTTP_OK; const
      StatusMessage: String='')
```

Available In: Server Applications

Sends a response for a HEAD request.

THTTPServerRequest.SendContentStream Method

```
procedure SendContentStream(ContentStream: TStream; StatusCode:
      Integer=HTTP_OK; const StatusMessage: String='')
```

Available In: Server Applications

Sends a UTF-8-encoded text stream response with a **Content-Type** header of "text/html; charset=utf-8" along with an optional status code and message.

THTTPServerRequest.SendCustomContent Method

```
procedure SendCustomContent(const Content: String; const
      ContentType: String; const ContentEncoding: String=''; const
      ContentDisposition: String='')
```

Available In: Server Applications

Sends a UTF-8-encoded text response with a custom content type, encoding, and disposition.

THTTPServerRequest.SendCustomContentHeader Method

```
procedure SendCustomContentHeader(const ContentType: String;
      const ContentEncoding: String=''; const ContentDisposition:
      String='')
```

Available In: Server Applications

Sends a custom content response for a HEAD request.

THTTPServerRequest.SendCustomContentStream Method

```
procedure SendCustomContentStream(ContentStream: TStream; const
      ContentType: String; const ContentEncoding: String=''; const
      ContentDisposition: String='')
```

Available In: Server Applications

Sends a binary stream response with a custom content type, encoding, and disposition.

THTTPServerRequest.SendError Method

```
procedure SendError(StatusCode: Integer; const StatusMessage:
      String; const Content: String='')
```

Available In: Server Applications

Sends a status code and message along with an optional UTF-8-encoded text response with a **Content-Type** header of "text/plain; charset=utf-8".

THTTPServerRequest.SendRedirect Method

```
procedure SendRedirect(const NewLocationURL: String; const
      Content: String=''; StatusCode: Integer=HTTP_FOUND; const
      StatusMessage: String='Redirecting')
```

Available In: Server Applications

Sends a redirect response to a new URL along with an optional UTF-8-encoded text response with a **Content-Type** header of "text/html; charset=utf-8". By default, the redirect HTTP status code is 302, but you can optionally specify a different status code.

10.120 TIcon Component

Unit: WebIcons

Inherits From TIconControl

Available In: Visual Client Applications

The TIcon component represents an icon control. An icon control displays an icon, referenced by the Icon property, that contains a single background image or a single font icon. Icons are defined and stored in the icon library.

Properties	Methods	Events
Cursor		OnAnimationComplete
Hint		OnAnimationsComplete
Icon		OnCaptureEnd
Opacity		OnCaptureStart
		OnCapturing
		OnClick
		OnContextMenu
		OnDblClick
		OnHide
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TIcon.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TIcon.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TIcon.Icon Property

```
property Icon: TIconProperties
```

Available In: Visual Client Applications

Specifies the properties of the icon used with the control.

TIcon.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TIcon.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TIcon.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TIcon.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TIcon.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TIcon.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TIcon.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TIcon.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TIcon.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TIcon.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TIcon.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TIcon.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TIcon.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TIcon.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TIcon.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TIcon.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TIcon.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TIcon.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TIcon.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TIcon.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TIcon.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TIcon.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.121 TIconAnimation Component

Unit: WebCtrls

Inherits From TComponent

Available In: Visual Client Applications

The TIconAnimation class represents the animation attributes to use for animating a specific property of icons. This class is used with various controls to specify the rotation animation properties of the icon used with the control.

Properties	Methods	Events
Duration		
Style		

TIconAnimation.Duration Property

```
property Duration: Integer
```

Available In: Visual Client Applications

Specifies how long, in milliseconds, the animation should take to execute.

TIconAnimation.Style Property

```
property Style: TAnimationStyle
```

Available In: Visual Client Applications

Specifies the style of the animation, which controls how the animation transforms a given UI element/control property. Currently, the supported styles include all of the standard easing transformations (including linear).

10.122 TIconButton Component

Unit: WebBtns

Inherits From TRepeatControl

Available In: Visual Client Applications

The TIconButton component represents an icon button control. An icon button control displays an icon, referenced by the Icon property, as a button that can be pressed. Icons are stored in the icon library.

Properties	Methods	Events
Cursor		OnAnimationComplete
Enabled		OnAnimationsComplete
Hint		OnCaptureEnd
Icon		OnCaptureStart
RepeatClick		OnCapturing
RepeatClickInterval		OnClick
		OnContextMenu
		OnHide
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TIconButton.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TIconButton.Enabled Property

```
property Enabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the control is enabled or disabled. When a control is disabled, it is displayed in a disabled state. The default value is True.

TIconButton.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TIconButton.Icon Property

```
property Icon: TIconProperties
```

Available In: Visual Client Applications

Specifies the properties of the icon used with the control.

TIconButton.RepeatClick Property

```
property RepeatClick: Boolean
```

Available In: Visual Client Applications

Specifies whether the OnClick event handler should be triggered every RepeatClickInterval milliseconds while the button is pressed.

TIconButton.RepeatClickInterval Property

```
property RepeatClickInterval: Integer
```

Available In: Visual Client Applications

Specifies the interval, in milliseconds, to trigger the OnClick event handler when the RepeatClick is True and the button is pressed.

TIconButton.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TIconButton.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TIconButton.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TIconButton.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TIconButton.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TIconButton.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TIconButton.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TIconButton.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TIconButton.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TIconButton.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TIconButton.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TIconButton.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TIconButton.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TIconButton.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TIconButton.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TIconButton.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TIconButton.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TIconButton.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TIconButton.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TIconButton.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.123 TIconControl Component

Unit: WebIcons

Inherits From TControl

Available In: Visual Client Applications

The TIconControl control is the base class for icon controls, and contains all of the core icon control functionality in the form of public methods and protected properties/events that descendant classes can use to create customized icon controls.

Properties	Methods	Events

10.124 TIconLibrary Component

Unit: WebUI

Inherits From TComponent

Available In: Visual Client Applications

The TIconLibrary class represents an icon library. An icon library is a list of embedded icon images stored as background images in control interface states. Each icon has a unique name and can be applied to a UI element so that the element displays the icon as its background image.

Properties	Methods	Events
	ApplyIcon	
	GetIconNames	

TIconLibrary.ApplyIcon Method

```
function ApplyIcon(AElement: TElement; const AIconName: String):
      Boolean
```

Available In: Visual Client Applications

Use this method to apply an icon as the background image of an element. The AElement parameter specifies the element on which to apply the icon, and the AIconName parameter specifies the name of the icon to apply.

TIconLibrary.GetIconNames Method

```
function GetIconNames: array of String
```

Available In: Visual Client Applications

Use this method to get a list of icon names available in the icon library as an array of strings.

10.125 TIconProperties Component

Unit: WebCtrls

Inherits From TComponent

Available In: Visual Client Applications

The TIconProperties class represents the icon properties of icons that are embedded in various controls.

Properties	Methods	Events
Animating	StartAnimation	
Animation	StopAnimation	
FontIcon		
Height		
IconName		
Width		

TIconProperties.Animating Property

```
property Animating: Boolean
```

Available In: Visual Client Applications

Indicates whether the icon's rotation animation is executing.

TIconProperties.Animation Property

```
property Animation: TIconAnimation
```

Available In: Visual Client Applications

Specifies the rotation animation properties for the icon.

TIconProperties.FontIcon Property

```
property FontIcon: TFontIcon
```

Available In: Visual Client Applications

Specifies the properties of the font icon (if a font icon is being used with the icon). These properties default to the pre-defined font icon properties of the icon specified by the IconName property, and may change when the IconName property is changed.

TIconProperties.Height Property

```
property Height: Integer
```

Available In: Visual Client Applications

Specifies the height of the icon. This value defaults to the pre-defined height of the icon specified by the IconName property, and may change when the IconName property is changed.

TIconProperties.IconName Property

```
property IconName: String
```

Available In: Visual Client Applications

Specifies the name of the icon to use with the control.

TIconProperties.Width Property

```
property Width: Integer
```

Available In: Visual Client Applications

Specifies the width of the icon. This value defaults to the pre-defined width of the icon specified by the IconName property, and may change when the IconName property is changed.

TIconProperties.StartAnimation Method

```
procedure StartAnimation
```

Available In: Visual Client Applications

Starts the rotation animation for the icon.

TIconProperties.StopAnimation Method

```
procedure StopAnimation
```

Available In: Visual Client Applications

Stops the rotation animation for the icon.

10.126 TImage Component

Unit: WebBrwsr

Inherits From TWebControl

Available In: Visual Client Applications

The TImage component represents an image control. An image control displays an image specified by the resource URL property at run-time.

> **Note**
> This control does not provide support for displaying images at design-time. If you wish to do so, you should specify the background image for a control. Only background images can be embedded in visual client application.

Properties	Methods	Events
ActualHeight		OnAnimationComplete
ActualWidth		OnAnimationsComplete
Background		OnCaptureEnd
Border		OnCaptureStart
ContentLayout		OnCapturing
Corners		OnClick
Cursor		OnContextMenu
DataColumn		OnDblClick
DataSet		OnError
Hint		OnHide
InsetShadow		OnLoad
Loaded		OnMouseDown
Opacity		OnMouseEnter
OutsetShadow		OnMouseLeave
Padding		OnMouseMove
URL		OnMouseUp
		OnMouseWheel
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove

		OnTouchStart
		OnUnload

TImage.ActualHeight Property

```
property ActualHeight: Integer
```

Available In: Visual Client Applications

Indicates the actual height of the loaded image when the URL property is specified and the Loaded property is True.

An event handler can be attached to the OnLoad event to execute code when the image is loaded.

TImage.ActualWidth Property

```
property ActualWidth: Integer
```

Available In: Visual Client Applications

Indicates the actual width of the loaded image when the URL property is specified and the Loaded property is True.

An event handler can be attached to the OnLoad event to execute code when the image is loaded.

TImage.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background for the control.

TImage.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TImage.ContentLayout Property

```
property ContentLayout: TContentLayout
```

Available In: Visual Client Applications

Specifies the layout properties of the image content contained within the control.

TImage.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TImage.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TImage.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TImage.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

TImage.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TImage.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Available In: Visual Client Applications

Specifies the inset shadow for the control.

TImage.Loaded Property

```
property Loaded: Boolean
```

Available In: Visual Client Applications

Indicates whether the image specified by the URL property has been loaded.

An event handler can be attached to the OnLoad event to execute code when the image is loaded.

TImage.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TImage.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Available In: Visual Client Applications

Specifies the outset shadow for the control.

TImage.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

TImage.URL Property

```
property URL: String
```

Available In: Visual Client Applications

Specifies the URL for the image. Whenever the URL property changes, the OnUnload event is triggered immediately. The OnLoad event is triggered once the image has been loaded.

TImage.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TImage.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TImage.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TImage.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TImage.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TImage.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TImage.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TImage.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TImage.OnError Event

```
property OnError: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the download of the image encounters an error.

TImage.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TImage.OnLoad Event

```
property OnLoad: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the image specified by the URL property has been completely loaded.

TImage.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TImage.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TImage.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TImage.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TImage.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TImage.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

Available In: Visual Client Applications

This event is triggered whenever the mouse wheel is rotated forward or backward. Return True from the event handler to indicate that the event was handled and should not continue to propagate to parent controls.

TImage.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TImage.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TImage.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TImage.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TImage.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TImage.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TImage.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

TImage.OnUnload Event

```
property OnUnload: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the currently-loaded image specified by the URL property has been unloaded.

10.127 TImageElement Component

Unit: WebUI

Inherits From TWebElement

Available In: Visual Client Applications

The TImageElement class is the element class for image elements, and contains all of the image functionality in the form of public methods and properties/events that control classes can use to create image controls.

> **Note**
> This element does not provide support for images at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events
ActualHeight		
ActualWidth		

TImageElement.ActualHeight Property

```
property ActualHeight: Integer
```

Available In: Visual Client Applications

Indicates the actual height of the loaded image when the URL property is specified and the Loaded property is True.

An event handler can be attached to the OnLoad event to execute code when the image is loaded.

TImageElement.ActualWidth Property

```
property ActualWidth: Integer
```

Available In: Visual Client Applications

Indicates the actual width of the loaded image when the URL property is specified and the Loaded property is True.

An event handler can be attached to the OnLoad event to execute code when the image is loaded.

10.128 TInputControl Component

Unit: WebCtrls

Inherits From TBindableColumnControl

Available In: Visual Client Applications

The TInputControl control is the base class for input controls, and contains all of the core input functionality in the form of public methods and protected properties/events that descendant classes can use to create customized input controls.

Properties	Methods	Events
Error		
InputID		

TInputControl.Error Property

```
property Error: Boolean
```

Available In: Visual Client Applications

Specifies whether the input control contains an invalid input value. Setting this property to True will cause the interface state of the control to change to an "Error" state to alert the user that the input value is invalid.

TInputControl.InputID Property

```
property InputID: String
```

Available In: Visual Client Applications

Specifies the unique DOM ID to assign to the control's input element. This is useful for situations where you need to identify the element from external Javascript code. By default, the interface manager never assigns DOM IDs to elements because it doesn't need or use them to identify UI elements.

Note
This is different from the TControl ClientID property, which is used for accessing the outer client DOM element for a control. An input control typically has both a client element and an input element.

10.129 TInputElement Component

Unit: WebUI

Inherits From TElement

Available In: Visual Client Applications

The TInputElement class is the base element class for input elements, and contains all of the input functionality in the form of public methods and properties/events that control classes can use to create input controls.

> **Note**
> This element does not provide support for input elements at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events
AutoComplete	SelectAll	
FieldName	SelectNone	
InputValue		
MaxLength		
Placeholder		
ReadOnly		
SelectionEnd		
SelectionStart		
SpellCheck		

TInputElement.AutoComplete Property

```
property AutoComplete: TAutoCompleteType
```

Available In: Visual Client Applications

Specifies how to handle auto-completion for the input element. Auto-completion allows the browser to display a list of suggestions for input values, based upon earlier input values entered by the user.

TInputElement.FieldName Property

```
property FieldName: String
```

Available In: Visual Client Applications

Specifies the HTML form value name for the input element. This name is used in conjunction with the InputValue property to generate the HTML form values data used when submitting HTML forms to a web server.

TInputElement.InputValue Property

```
property InputValue: String
```

Available In: Visual Client Applications

Specifies the current value of the input element as a string.

TInputElement.MaxLength Property

```
property MaxLength: Integer
```

Available In: Visual Client Applications

Specifies the maximum allowable length, in characters, of the InputValue property for the element. A value of 0 specifies an unlimited allowable length.

TInputElement.Placeholder Property

```
property Placeholder: String
```

Available In: Visual Client Applications

Specifies a hint to display in the input element before the InputValue property has been assigned a non-blank value.

TInputElement.ReadOnly Property

```
property ReadOnly: Boolean
```

Available In: Visual Client Applications

Specifies whether the input element can be modified by the user. Input elements can always be programmatically modified.

TInputElement.SelectionEnd Property

```
property SelectionEnd: Integer
```

Available In: Visual Client Applications

Specifies the ending position (1-based) of the selected characters in the input value. For example, if the element contains the input value "Hello World", setting the SelectionStart property to 7 and the SelectionEnd property to 11 will result in the text "World" being selected.

TInputElement.SelectionStart Property

```
property SelectionStart: Integer
```

Available In: Visual Client Applications

Specifies the starting position (1-based) of the selected characters in the input value. For example, if the element contains the input value "Hello World", setting the SelectionStart property to 7 and the SelectionEnd property to 11 will result in the text "World" being selected.

TInputElement.SpellCheck Property

```
property SpellCheck: Boolean
```

Available In: Visual Client Applications

Specifies whether spell-checking will be enabled for the input element.

TInputElement.SelectAll Method

```
procedure SelectAll
```

Available In: Visual Client Applications

Use this method to change the active selection for the input element so that all of the characters in the InputValue property are selected.

TInputElement.SelectNone Method

```
procedure SelectNone
```

Available In: Visual Client Applications

Use this method to change the active selection for the input element so that none of the characters in the InputValue property are selected.

10.130 TInsetShadow Component

Unit: WebUI

Inherits From TShadow

Available In: Visual Client Applications

The TInsetShadow class represents the inset shadow of a UI element or control. The inset shadow appears within the bounds of the client rectangle for the element.

Properties	Methods	Events

10.131 TInterface Component

Unit: WebUI

Inherits From TPersistent

Available In: Visual Client Applications

The TInterface class represents a control interface that has been loaded by the interface manager at run-time.

Properties	Methods	Events
ControlClassName	Create	
States		

TInterface.ControlClassName Property

```
property ControlClassName: String
```

Available In: Visual Client Applications

Specifies the class name of the control that will use the control interface.

TInterface.States Property

```
property States: TInterfaceStates
```

Available In: Visual Client Applications

Specifies the various states the make up the control interface.

TInterface.Create Method

```
constructor Create(const AControlClassName: String='')
```

Available In: Visual Client Applications

Use this method to create a new instance of the TInterface class. The AControlClassName parameter indicates the class name of the control that will use the control interface.

10.132 TInterfaceController Component

Unit: WebUI

Inherits From TComponent

Available In: Visual Client Applications

The TInterfaceController class represents an element's controller. A controller instance is assigned to any element that serves as the base element for a TInterfaceController class descendant.

> **Note**
> The TInterfaceController class is never instantiated directly. It is used only as a bridge component between the TElement class and the TControl class.

Properties	Methods	Events
InterfaceClassName	RefreshInterface	
InterfaceState	ResetInterface	

TInterfaceController.InterfaceClassName Property

```
property InterfaceClassName: String
```

Available In: Visual Client Applications

Indicates the interface class name for the controller. The interface class name is used to determine which control interface to apply to the controller as the InterfaceState property changes.

TInterfaceController.InterfaceState Property

```
property InterfaceState: String
```

Available In: Visual Client Applications

Gets or sets the current interface state for the controller. The valid interface states for a controller are implementation-defined within the control interface that applies to the controller, which is determined by the InterfaceClassName property.

TInterfaceController.RefreshInterface Method

```
procedure RefreshInterface
```

Available In: Visual Client Applications

Use this method to refresh the control interface for the controller. Refreshing a control interface causes the current interface state to be re-applied to the controller from a control interface.

TInterfaceController.ResetInterface Method

```
procedure ResetInterface
```

Available In: Visual Client Applications

Use this method to reset the control interface for the controller. Resetting a control interface causes the controller to be initialized using the current interface state from a control interface.

10.133 TInterfaceManager Component

Unit: WebUI

Inherits From TObject

Available In: Visual Client Applications

The TInterfaceManager class represents the interface manager, which is responsible for managing the user interface of an application at both design-time and run-time. A global instance of the TInterfaceManager class is created automatically at application startup, and is called InterfaceManager.

> **Note**
> Do not create any instances of this class manually, and always use the global instance in order to avoid any conflicts or issues.

Properties	Methods	Events
ActiveElement	ApplyInterface	OnError
ApplicationTitle	BeginAnimation	OnIdle
IdleTimeout	CancelAnimation	OnViewportResize
Interfaces	ContentHeight	OnViewportScroll
IsAndroid	ContentWidth	
IsIOS	ContinueAnimation	
IsWindowsPhone	Create	
RootElement	CreateElement	
ViewportHeight	CreateTimeout	
ViewportResizeDelay	CreateTimer	
ViewportScrollLeft	FreeTimeout	
ViewportScrollTop	FreeTimer	
ViewportWidth	GetInterfaceStateNames	
	GetMetaContent	
	GetResource	
	GetResourceCompressed	
	ViewportScroll	

TInterfaceManager.ActiveElement Property

```
property ActiveElement: TElement
```

Available In: Visual Client Applications

Indicates the active element in the user interface. The active UI element is the element that currently has focus.

TInterfaceManager.ApplicationTitle Property

```
property ApplicationTitle: String
```

Available In: Visual Client Applications

Indicates the title of the application, which is the descriptive name that appears in the web browser for the application's tab or page.

TInterfaceManager.IdleTimeout Property

```
property IdleTimeout: Integer
```

Available In: Visual Client Applications

Specifies the time, in seconds, that the application should wait on user input (keypresses, mouse clicks, or touches) before triggering the OnIdle event. This is useful for functionality such as making sure that any authentication information cached for the current user is discarded after a certain period of inactivity, thus forcing the user to login again when interaction with the application is resumed.

Note
Mouse movement alone is not enough to reset the idle timeout. The user must specifically press a key or mouse button, or touch the surface of the screen.

TInterfaceManager.Interfaces Property

```
property Interfaces: TInterfaces
```

Available In: Visual Client Applications

Indicates the control interfaces that are loaded by the interface manager at runtime. These control interfaces can be modified by the application at runtime in order to affect the visual appearance of controls.

TInterfaceManager.IsAndroid Property

```
property IsAndroid: Boolean
```

Available In: Visual Client Applications

Indicates whether the platform running the application is the Android platform.

TInterfaceManager.IsIOS Property

```
property IsIOS: Boolean
```

Available In: Visual Client Applications

Indicates whether the platform running the application is the IOS platform.

TInterfaceManager.IsWindowsPhone Property

```
property IsWindowsPhone: Boolean
```

Available In: Visual Client Applications

Indicates whether the platform running the application is the Windows Phone platform.

TInterfaceManager.RootElement Property

```
property RootElement: TElement
```

Available In: Visual Client Applications

Indicates the root element of the user interface. At design-time, the root element may be one of two things, depending upon whether the form designer or control interface editor is active:

- If the form designer is active, then the root element will be the base element associated with the active TFormControl instance.
- If the control interface editor is active, then the root element will be the base element for the currently-selected control interface state.

At run-time, the root element is always the base element for the global Application Surface instance.

TInterfaceManager.ViewportHeight Property

```
property ViewportHeight: Integer
```

Available In: Visual Client Applications

Indicates the height of the browser viewport.

TInterfaceManager.ViewportResizeDelay Property

```
property ViewportResizeDelay: Integer
```

Available In: Visual Client Applications

Specifies how long, in milliseconds, the interface manager will wait after a browser viewport resize before updating the user interface.

TInterfaceManager.ViewportScrollLeft Property

```
property ViewportScrollLeft: Integer
```

Available In: Visual Client Applications

Indicates the amount, in pixels, that the browser viewport has been scrolled to the right.

Specify a new value to manually scroll the browser viewport to the left or right. A value of 0 means that the viewport is scrolled all the way to its left-most position.

TInterfaceManager.ViewportScrollTop Property

```
property ViewportScrollTop: Integer
```

Available In: Visual Client Applications

Indicates the amount, in pixels, that the browser viewport has been scrolled towards the bottom.

Specify a new value to manually scroll the browser viewport towards the top or bottom. A value of 0 means that the viewport is scrolled all the way to its top-most position.

TInterfaceManager.ViewportWidth Property

```
property ViewportWidth: Integer
```

Available In: Visual Client Applications

Indicates the width of the browser viewport.

TInterfaceManager.ApplyInterface Method

```
function ApplyInterface(AElement: TElement; const AClassName:
      String; const AState: String; Initializing: Boolean=False;
      Resetting: Boolean=False): Boolean
```

Available In: Visual Client Applications

Use this method to apply a control interface to a base UI element. The AElement parameter specifies the base UI element instance, the AClassName parameter specifies the name of the control interface's class name, the AState parameter specifies the control interface state to apply, and the optional Initializing parameter specifies whether the control interface state should be applied (Initializing=False) or whether the base element should be initialized using the control interface state.

TInterfaceManager.BeginAnimation Method

```
function BeginAnimation(AHandler: TInterfaceAnimationEvent):
      Integer
```

Available In: Visual Client Applications

Use this method to add an animation and register an animation frame event handler to be called based upon the monitor refresh rate, if supported, or at 30 times per second if the browser doesn't support the first method.

Animations require that each animation frame event handler execution be scheduled using the BeginAnimation or ContinueAnimation methods, with the BeginAnimation call required at the start of the animation, and the ContinueAnimation call required for all subsequent animation frames until the CancelAnimation method is called.

The return value is an integer that represents the animation, and can be used in further animation operations using the ContinueAnimation and CancelAnimation methods.

Note
This method is not available at design-time. Any calls to it will cause a design-time compilation error when compiling the component library.

TInterfaceManager.CancelAnimation Method

```
procedure CancelAnimation(AID: Integer)
```

Available In: Visual Client Applications

Use this method to stop an animation that was started using the BeginAnimation method, or continued using the ContinueAnimation method.

> **Note**
> This method is not available at design-time. Any calls to it will cause a design-time compilation error when compiling the component library.

TInterfaceManager.ContentHeight Method

```
function ContentHeight(AFont: TFont; const AContent: String;
      AHTMLContent: Boolean=False; AAlignment:
      TContentAlignment=caLeft; ADirection:
      TContentDirection=cdLeftToRight; AWrap: Boolean=False;
      AWrapWidth: Integer=0): Integer
```

Available In: Visual Client Applications

Use this method to measure the height of text content using the specified font and formatting parameters.

TInterfaceManager.ContentWidth Method

```
function ContentWidth(AFont: TFont; const AContent: String;
      AHTMLContent: Boolean=False; AAlignment:
      TContentAlignment=caLeft; ADirection:
      TContentDirection=cdLeftToRight; AWrap: Boolean=False;
      AWrapWidth: Integer=0): Integer
```

Available In: Visual Client Applications

Use this method to measure the width of text content using the specified font and formatting parameters.

TInterfaceManager.ContinueAnimation Method

```
function ContinueAnimation(AID: Integer; AHandler:
      TInterfaceAnimationEvent): Integer
```

Available In: Visual Client Applications

Use this method to re-schedule an animation that was started using the BeginAnimation method or already continued using this method.

Animations require that each animation frame event handler execution be scheduled using the BeginAnimation or ContinueAnimation methods, with the BeginAnimation call required at the start of the animation, and the ContinueAnimation call required for all subsequent animation frames until the CancelAnimation method is called.

Warning
This method should be called from within the animation event handler specified by the last BeginAnimation or ContinueAnimation call.

Note
This method is not available at design-time. Any calls to it will cause a design-time compilation error when compiling the component library.

TInterfaceManager.Create Method

```
constructor Create
```

Available In: Visual Client Applications

Use this method to create a new instance of the TInterfaceManager class.

TInterfaceManager.CreateElement Method

```
function CreateElement(const AName: String; AParent:
      TElement=nil; const AClassName: String=ELEMENT_CLASS_DIV;
      AContainer: Boolean=False; AEvents: Boolean=False; ADynamic:
      Boolean=False): TElement
```

Available In: Visual Client Applications

Use this method to create a new UI element.

The AName parameter specifies the name of the element.

The optional AParent parameter specifies the parent element, if any.

The optional AClassName parameter specifies the element class name to use when creating the element.

The optional AContainer parameter specifes whether or not the element is a container. A container element is one that is capable of being a container for other elements at design-time.

The optional AEvents parameter specifies whether the element wants design-time mouse events in order to allow the developer to interact with the element at design-time.

The optional ADynamic parameter specifies whether the element is being instantiated dynamically at design-time via interaction by the developer.

TInterfaceManager.CreateTimeout Method

```
function CreateTimeout(AHandler: TInterfaceTimeoutEvent;
      AInterval: Integer): Integer
```

Available In: Visual Client Applications

Use this method to create a timeout event. The AHandler parameter specifies the event handler to call when the AInterval milliseconds parameter has elapsed.

TInterfaceManager.CreateTimer Method

```
function CreateTimer(AHandler: TInterfaceTimerEvent; AInterval:
      Integer): Integer
```

Available In: Visual Client Applications

Use this method to create a timer event. The AHandler parameter specifies the event handler to call every time the AInterval milliseconds parameter has elapsed.

TInterfaceManager.FreeTimeout Method

```
procedure FreeTimeout(AID: Integer)
```

Available In: Visual Client Applications

Use this method to free a timeout event.

TInterfaceManager.FreeTimer Method

```
procedure FreeTimer(AID: Integer)
```

Available In: Visual Client Applications

Use this method to free a timer event.

TInterfaceManager.GetInterfaceStateNames Method

```
function GetInterfaceStateNames(const AClassName: String): array
      of String
```

Available In: Visual Client Applications

Use this method to retrieve a list of defined state names as an array of strings. The AClassName is the control interface class name for which you want the list of state names.

> **Note**
> This method is used by the icon library to retrieve the list of icons defined in the icon library.

TInterfaceManager.GetMetaContent Method

```
function GetMetaContent(const Name: String): String
```

Available In: Visual Client Applications

Use this method to retrieve embedded meta content (application title, version, and other properties) for an application at run-time.

> **Note**
> This method is not available at design-time. Any calls to it will cause a design-time compilation error when compiling the component library.

TInterfaceManager.GetResource Method

```
function GetResource(const ResourceType: String; const
      ResourceName: String): String
```

Available In: Visual Client Applications

Use this method to retrieve an embedded resource for an application at run-time.

> **Note**
> This method is not available at design-time. Any calls to it will cause a design-time compilation error when compiling the component library.

TInterfaceManager.GetResourceCompressed Method

```
function GetResourceCompressed(const ResourceType: String; const
      ResourceName: String): Boolean
```

Available In: Visual Client Applications

Use this method to retrieve a compressed and embedded resource for an application at run-time.

> **Note**
> This method is not available at design-time. Any calls to it will cause a design-time compilation error when compiling the component library.

TInterfaceManager.ViewportScroll Method

```
procedure ViewportScroll(X,Y: Integer)
```

Available In: Visual Client Applications

Use this method to manually scroll the browser viewport at run-time.

> **Note**
> This method is not available at design-time. Any calls to it will cause a design-time compilation error when compiling the component library.

TInterfaceManager.OnError Event

```
property OnError: TInterfaceErrorEvent
```

Available In: Visual Client Applications

This event is triggered when any unhandled exception occurs in the application at runtime in the browser.

Note
Do not set an event handler for this event. The global TApplication instance for visual applications assigns an event handler for this event.

TInterfaceManager.OnIdle Event

```
property OnIdle: TInterfaceIdleEvent
```

Available In: Visual Client Applications

This event is triggered when the application's IdleTimeout property has been exceeded.

Note
Do not set an event handler for this event. The global TApplication instance for visual applications assigns an event handler for this event.

TInterfaceManager.OnViewportResize Event

```
property OnViewportResize: TInterfaceViewportResizeEvent
```

Available In: Visual Client Applications

This event is triggered whenever the browser viewport's width and/or height are changed.

TInterfaceManager.OnViewportScroll Event

```
property OnViewportScroll: TInterfaceViewportScrollEvent
```

Available In: Visual Client Applications

This event is triggered whenever the browser viewport is scrolled horizontally or vertically.

10.134 TInterfaces Component

Unit: WebUI

Inherits From TObject

Available In: Visual Client Applications

The TInterfaces class represents the control interfaces that have been loaded by the interface manager at run-time.

> **Note**
> You can modify the control interfaces after they have been loaded at runtime in order to customize how various controls look.

Properties	Methods	Events
Count	Create	
Interfaces	FindInterfaceByClassName	
	Load	
	RemoveAll	

TInterfaces.Count Property

```
property Count: Integer
```

Available In: Visual Client Applications

Indicates the total number of defined control interfaces.

TInterfaces.Interfaces Property

```
property Interfaces[Index: Integer]: TInterface
```

Available In: Visual Client Applications

Accesses a control interface by its index position in the defined control interfaces.

TInterfaces.Create Method

```
constructor Create
```

Available In: Visual Client Applications

Use this method to create a new instance of the TInterfaces class.

TInterfaces.FindInterfaceByClassName Method

```
function FindInterfaceByClassName(const AClassName: String):
      TInterface
```

Available In: Visual Client Applications

Use this method to find a control interface by its control class name. If the control interface does not exist, then nil will be returned.

TInterfaces.Load Method

```
procedure Load
```

Available In: Visual Client Applications

Use this method to load all control interfaces bundled with the visual application from the HTML loader file for the application. This method is automatically called during application startup and should not be called manually.

TInterfaces.RemoveAll Method

```
procedure RemoveAll
```

Available In: Visual Client Applications

Use this method to remove all loaded control interfaces. It is not advised to use this method because doing so can cause a visual application to lose all interface attributes.

10.135 TInterfaceState Component

Unit: WebUI

Inherits From TPersistent

Available In: Visual Client Applications

The TInterfaceState class represents a state defined for a control interface that has been loaded by the interface manager at run-time.

Properties	Methods	Events
Name		
RootElement		

TInterfaceState.Name Property

```
property Name: String
```

Available In: Visual Client Applications

Specifies the name of the control interface state.

TInterfaceState.RootElement Property

```
property RootElement: TElement
```

Available In: Visual Client Applications

Indicates the root element for the control interface state.

10.136 TInterfaceStates Component

Unit: WebUI

Inherits From TPersistent

Available In: Visual Client Applications

The TInterfaceStates class represents the states defined for a control interface that has been loaded by the interface manager at run-time.

Properties	Methods	Events
Count	Create	
DefaultState	FindState	
State	NewState	
	RemoveState	
	RenameState	

TInterfaceStates.Count Property

```
property Count: Integer
```

Available In: Visual Client Applications

Indicates the total number of defined control interface states.

TInterfaceStates.DefaultState Property

```
property DefaultState: TInterfaceState
```

Available In: Visual Client Applications

Indicates the default control interface state, which corresponds to the first defined control interface state.

TInterfaceStates.State Property

```
property State[Index: Integer]: TInterfaceState
```

Available In: Visual Client Applications

Accesses a control interface state by its index position in the defined control interface states.

TInterfaceStates.Create Method

```
constructor Create(AInterface: TInterface)
```

Available In: Visual Client Applications

Use this method to create a new instance of the TInterfaceStates class. The AInterface parameter indicates the parent interface instance that will manage the states.

TInterfaceStates.FindState Method

```
function FindState(const AName: String): TInterfaceState
```

Available In: Visual Client Applications

Use this method to find a control interface state by its name. If the state does not exist, then nil will be returned.

TInterfaceStates.NewState Method

```
function NewState(const AName: String; const FromName:
      String=''): TInterfaceState
```

Available In: Visual Client Applications

Use this method to create a new control interface state. The AName parameter indicates the name of the new state, and the optional FromName parameter indicates the name of an existing state that will be used to initialize the new state. If the state indicated by the AName parameter already exists, then an exception will be raised.

TInterfaceStates.RemoveState Method

```
function RemoveState(const AName: String): Boolean
```

Available In: Visual Client Applications

Use this method to remove a control interface state. If the state does not exist, then False will be returned.

TInterfaceStates.RenameState Method

```
function RenameState(const AName: String; const NewName:
      String): Boolean
```

Available In: Visual Client Applications

Use this method to rename a control interface state. If the state does not exist, then False will be returned.

10.137 TLabel Component

Unit: WebLabels

Inherits From TLabelControl

Available In: Visual Client Applications

The TLabel component represents a label control. A label control displays text content using a specific font and formatting, including alignment and wrapping.

Properties	Methods	Events
AllowCopy		OnAnimationComplete
AutoSize		OnAnimationsComplete
Background		OnCaptureEnd
Border		OnCaptureStart
Caption		OnCapturing
Corners		OnClick
Cursor		OnContextMenu
DataColumn		OnDblClick
DataSet		OnHide
FocusControl		OnMouseDown
Font		OnMouseEnter
Format		OnMouseLeave
Hint		OnMouseMove
Opacity		OnMouseUp
OutsetShadow		OnMove
Padding		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TLabel.AllowCopy Property

```
property AllowCopy: Boolean
```

Available In: Visual Client Applications

Specifies whether the label control will allow its contents to be copied by the user.

Note
You must also set the Cursor property to crText if you want the user to be able to directly copy the contents of the label using the mouse or touch.

TLabel.AutoSize Property

```
property AutoSize: TAutoSize
```

Available In: Visual Client Applications

Specifies how (if at all) the control should automatically be sized based upon the Caption and Format properties.

TLabel.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

TLabel.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TLabel.Caption Property

```
property Caption: TCaption
```

Available In: Visual Client Applications

Specifies the textual caption to display in the control. The default value is ''.

TLabel.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TLabel.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TLabel.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TLabel.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

TLabel.FocusControl Property

```
property FocusControl: TInputControl
```

Available In: Visual Client Applications

Specifies a control that will receive focus when the label is clicked.

TLabel.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TLabel.Format Property

```
property Format: TFormat
```

Available In: Visual Client Applications

Specifies the content formatting to use for the control's Caption.

TLabel.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TLabel.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TLabel.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Available In: Visual Client Applications

Specifies the outset shadow for the control.

TLabel.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

TLabel.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TLabel.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TLabel.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TLabel.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TLabel.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TLabel.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TLabel.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TLabel.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TLabel.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TLabel.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TLabel.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TLabel.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TLabel.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TLabel.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TLabel.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TLabel.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TLabel.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TLabel.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TLabel.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TLabel.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TLabel.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.138 TLabelControl Component

Unit: WebLabels

Inherits From TBindableColumnControl

Available In: Visual Client Applications

The TLabelControl control is the base class for label controls, and contains all of the core label functionality in the form of public methods and protected properties/events that descendant classes can use to create customized label controls.

Properties	Methods	Events

10.139 TLayout Component

Unit: WebUI

Inherits From TElementAttribute

Available In: Visual Client Applications

The TLayout class represents the layout properties of a UI element or control. The layout properties include positioning, stretching, and layout space consumption. Please see the Layout Management topic for more information on how the layout properties are used.

Properties	Methods	Events
Consumption	SetToDefault	
Overflow		
Position		
Reset		
Stretch		

TLayout.Consumption Property

```
property Consumption: TLayoutConsumption
```

Available In: Visual Client Applications

Specifies how the UI element or control consumes space in the current layout rectangle, if at all.

TLayout.Overflow Property

```
property Overflow: TLayoutOverflow
```

Available In: Visual Client Applications

Specifies the direction in which the current UI element or control should reset the consumption of the prior UI element or control when the current UI element or control will not fit within the bounds of the current layout rectangle.

TLayout.Position Property

```
property Position: TLayoutPosition
```

Available In: Visual Client Applications

Specifies how the UI element or control is positioned in the current layout rectangle, if at all.

TLayout.Reset Property

```
property Reset: Boolean
```

Available In: Visual Client Applications

Specifies whether the UI element or control is resetting the space consumption direction in the current layout rectangle.

TLayout.Stretch Property

```
property Stretch: TLayoutStretch
```

Available In: Visual Client Applications

Specifies how the UI element or control is stretched in the current layout rectangle, if at all.

TLayout.SetToDefault Method

```
procedure SetToDefault
```

Available In: Visual Client Applications

Use this method to reset the layout's properties to their default values.

10.140 TLink Component

Unit: WebBrwsr

Inherits From TLinkControl

Available In: Visual Client Applications

The TLink component represents a link control. A link control displays link text using the Caption property that results in navigation to the specified URL property when the control is clicked at run-time.

> **Note**
> This control does not provide support for links at design-time.

Properties	Methods	Events
AutoSize		OnAnimationComplete
Background		OnAnimationsComplete
Border		OnCaptureEnd
Caption		OnCaptureStart
Corners		OnCapturing
Cursor		OnClick
DataColumn		OnContextMenu
DataSet		OnHide
Font		OnMouseDown
Format		OnMouseEnter
Hint		OnMouseLeave
NewWindow		OnMouseMove
Opacity		OnMouseUp
OutsetShadow		OnMove
Padding		OnShow
TabOrder		OnSize
TabStop		OnTouchCancel
URL		OnTouchEnd
		OnTouchMove
		OnTouchStart

TLink.AutoSize Property

```
property AutoSize: TAutoSize
```

Available In: Visual Client Applications

Specifies how (if at all) the control should automatically be sized based upon the Caption and Format properties.

TLink.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background for the control.

TLink.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TLink.Caption Property

```
property Caption: TCaption
```

Available In: Visual Client Applications

Specifies the textual caption to display in the control. The default value is ''.

TLink.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TLink.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TLink.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TLink.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

TLink.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TLink.Format Property

```
property Format: TFormat
```

Available In: Visual Client Applications

Specifies the content formatting to use for the control's Caption.

TLink.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TLink.NewWindow Property

```
property NewWindow: Boolean
```

Available In: Visual Client Applications

Specifies that the resource represented by the URL property should be opened in a new browser window or page when the link control is clicked.

TLink.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TLink.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Available In: Visual Client Applications

Specifies the outset shadow for the control.

TLink.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

TLink.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TLink.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TLink.URL Property

```
property URL: String
```

Available In: Visual Client Applications

Specifies the URL for the resource that is the target of the link.

TLink.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TLink.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TLink.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TLink.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TLink.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TLink.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TLink.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TLink.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TLink.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TLink.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TLink.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TLink.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TLink.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TLink.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TLink.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TLink.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TLink.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TLink.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TLink.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TLink.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.141 TLinkControl Component

Unit: WebBrwsr

Inherits From TBindableColumnControl

Available In: Visual Client Applications

The TLinkControl control is the base class for link controls, and contains all of the core link functionality in the form of public methods and protected properties/events that descendant classes can use to create customized link controls.

Properties	Methods	Events

10.142 TLinkElement Component

Unit: WebUI

Inherits From TElement

Available In: Visual Client Applications

The TLinkElement class is the element class for URL links, and contains all of the URL link functionality in the form of public methods and properties/events that control classes can use to create link controls.

> **Note**
> This element does not provide support for links at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events
NewWindow	OpenURL	
URL		

TLinkElement.NewWindow Property

```
property NewWindow: Boolean
```

Available In: Visual Client Applications

Specifies that the navigation that occurs in response to a click or selection should cause a new browser window to open.

TLinkElement.URL Property

```
property URL: String
```

Available In: Visual Client Applications

Specifies the URL of the link.

TLinkElement.OpenURL Method

```
procedure OpenURL
```

Available In: Visual Client Applications

Use this method to programmatically open the link specified in the URL property. If the NewWindow property is True, then the URL is opened in a new browser window or tab.

10.143 TListBox Component

Unit: WebLists

Inherits From TListControl

Available In: Visual Client Applications

The TListBox component represents a listbox control for displaying a list of selectable items specified by the Items property. Multiple items can be selected if the MultiSelect property is set to True.

> **Note**
> This control is a virtual control, meaning that it can store and display very large numbers of items efficiently by only using UI elements for the visible items in the control.

Properties	Methods	Events
AutoItemHeight		OnAnimationComplete
Corners		OnAnimationsComplete
Cursor		OnCaptureEnd
DataColumn		OnCaptureStart
DataSet		OnCapturing
Enabled		OnChange
Font		OnClick
Hint		OnContextMenu
ItemHeight		OnDblClick
ItemIndex		OnEnter
Items		OnExit
KeyPressInterval		OnHide
MultiSelect		OnKeyDown
ReadOnly		OnKeyPress
ScrollBar		OnKeyUp
ScrollSupport		OnMouseDown
SelectedCount		OnMouseEnter
Sorted		OnMouseLeave
TabOrder		OnMouseMove
TabStop		OnMouseUp
Text		OnMouseWheel
		OnMove
		OnScroll

		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchScroll
		OnTouchStart

TListBox.AutoItemHeight Property

```
property AutoItemHeight: Boolean
```

Available In: Visual Client Applications

Specifies that the displayed height of the items will automatically be set based upon the Font property settings. The default value is True.

TListBox.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TListBox.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TListBox.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TListBox.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

TListBox.Enabled Property

```
property Enabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TListBox.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TListBox.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TListBox.ItemHeight Property

```
property ItemHeight: Integer
```

Available In: Visual Client Applications

Specifies the height, in pixels, of the items displayed in the list control.

TListBox.ItemIndex Property

```
property ItemIndex: Integer
```

Available In: Visual Client Applications

Indicates the index of the currently-focused item in the list control, or -1 if no item is currently focused.

TListBox.Items Property

```
property Items: TStrings
```

Available In: Visual Client Applications

Specifies the items to display in the list control.

TListBox.KeyPressInterval Property

```
property KeyPressInterval: Integer
```

Available In: Visual Client Applications

Specifies the interval, in milliseconds, that is used by the control to combine user keystrokes into a search value that is then used for performing a near search on the Items property. Effectively, this means that the user has KeyPressInterval milliseconds in which to hit a key in order for the keystroke to be included as part of a near search. The default value is 300 milliseconds.

For example, if the user hits the "S", "M", and "I" keys within the KeyPressInterval property value, but hits the "T" key outside of the KeyPressInterval property, then the control will perform a near search using the value "SMI", followed by a near search using the value "T".

TListBox.MultiSelect Property

```
property MultiSelect: Boolean
```

Available In: Visual Client Applications

Specifies whether multiple items may be selected. If this property is True, then the user can select multiple items in the list control using an individual selection operation (Ctrl-Click), or using a range selection operation (Shift-Click). In addition, the developer can directly modify the Selected property to select or deselect any item(s) in the list control.

The default value is False.

TListBox.ReadOnly Property

```
property ReadOnly: Boolean
```

Available In: Visual Client Applications

Specifies whether the listbox's selected item(s) can be modified by the user. The default value is False.

Note
The listbox's selected item(s) can always be programmatically modified.

TListBox.ScrollBar Property

```
property ScrollBar: Boolean
```

Available In: Visual Client Applications

Specifies whether the vertical scrollbar should be shown for the listbox control.

Note
Even if this property is set to True, a vertical scrollbar will only be shown if the vertical size of the contents of the control exceed the client rectangle for the control.

TListBox.ScrollSupport Property

```
property ScrollSupport: Boolean
```

Available In: Visual Client Applications

Specifies whether to allow vertical scrolling in the listbox control.

Note
This property only applies to scrolling via touch and mouse wheel movements, and does not apply to programmatic scrolling or scrolling via the vertical scroll bar itself.

TListBox.SelectedCount Property

```
property SelectedCount: Integer
```

Available In: Visual Client Applications

Specifies the number of selected items in the list control. If the MultiSelect property is True, then this property can be used to find out how many items are selected. If the MultiSelect property is False, then this property will always be equal to 1 or 0 (if the list is empty).

TListBox.Sorted Property

```
property Sorted: Boolean
```

Available In: Visual Client Applications

Specifies whether the list items will automatically be sorted. The default value is False.

TListBox.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TListBox.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TListBox.Text Property

```
property Text: String
```

Available In: Visual Client Applications

Specifies the list control's selected items(s) as a string.

TListBox.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TListBox.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TListBox.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TListBox.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TListBox.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TListBox.OnChange Event

```
property OnChange: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TListBox.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TListBox.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TListBox.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TListBox.OnEnter Event

```
property OnEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control obtains input focus.

TListBox.OnExit Event

```
property OnExit: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control loses input focus.

TListBox.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TListBox.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses a key or key combination.

TListBox.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TListBox.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user releases a key or key combination.

TListBox.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TListBox.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TListBox.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TListBox.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TListBox.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TListBox.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

Available In: Visual Client Applications

This event is triggered whenever the mouse wheel is rotated forward or backward.

TListBox.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TListBox.OnScroll Event

```
property OnScroll: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever a scrollable control is scrolled horizontally or vertically.

TListBox.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TListBox.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TListBox.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TListBox.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TListBox.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TListBox.OnTouchScroll Event

```
property OnTouchScroll: TTouchScrollEvent
```

Available In: Visual Client Applications

This event is triggered whenever a touch moves in any direction over a touch-scroll-enabled control.

TListBox.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.144 TListControl Component

Unit: WebLists

Inherits From TInputControl

Available In: Visual Client Applications

The TListControl control is the base class for list controls, and contains all of the core list functionality in the form of public methods and protected properties/events that descendant classes can use to create customized list controls.

Properties	Methods	Events
	SelectAll	
	SelectRange	
	ToggleSelected	

TListControl.SelectAll Method

```
procedure SelectAll
```

Available In: Visual Client Applications

Use this method to select all items in the list control.

TListControl.SelectRange Method

```
procedure SelectRange(AFromIndex, AToIndex: Integer; AClear:
      Boolean=False)
```

Available In: Visual Client Applications

Use this method to select a range of items in the list control. The AClear parameter determines whether the existing set of selected items should be cleared before the new range of items is selected.

TListControl.ToggleSelected Method

```
procedure ToggleSelected(AIndex: Integer)
```

Available In: Visual Client Applications

Use this method to toggle the selection state of the item at the specified index.

10.145 TLocation Component

Unit: WebComps

Inherits From TObject

Available In: Client Applications

The TLocation class represents current location information for the global LocationServices instance of the TLocationServices class and contains properties for all available location information.

Properties	Methods	Events
Accuracy		
Altitude		
AltitudeAccuracy		
AltitudeAvailable		
Heading		
HeadingAvailable		
Latitude		
Longitude		
Speed		
SpeedAvailable		
Timestamp		

TLocation.Accuracy Property

```
property Accuracy: Double
```

Available In: Client Applications

Specifies the accuracy, in meters, of the Latitude and Longitude properties.

TLocation.Altitude Property

```
property Altitude: Double
```

Available In: Client Applications

Specifies the altitude, in meters, relative to sea level.

Note
This property is only valid when the AltitudeAvailable property is True.

TLocation.AltitudeAccuracy Property

```
property AltitudeAccuracy: Double
```

Available In: Client Applications

Specifies the accuracy, in meters, of the Altitude property.

Note
This property is only valid when the AltitudeAvailable property is True.

TLocation.AltitudeAvailable Property

```
property AltitudeAvailable: Boolean
```

Available In: Client Applications

Specifies whether altitude information was available as part of the returned location information.

Note
The Altitude and AltitudeAccuracy properties are only valid when the AltitudeAvailable property is True.

TLocation.Heading Property

```
property Heading: Double
```

Available In: Client Applications

Specifies the heading, in degrees, relative to true north. East is 90 degrees, south is 180 degrees, and west is 270 degrees. If the Speed property is 0, then this property will be 0.

> **Note**
> This property is only valid when the HeadingAvailable property is True.

TLocation.HeadingAvailable Property

```
property HeadingAvailable: Boolean
```

Available In: Client Applications

Specifies whether heading information was available as part of the returned location information.

> **Note**
> The Heading property is only valid when the HeadingAvailable property is True.

TLocation.Latitude Property

```
property Latitude: Double
```

Available In: Client Applications

Specifies the latitude, in meters.

TLocation.Longitude Property

```
property Longitude: Double
```

Available In: Client Applications

Specifies the longitude, in meters.

TLocation.Speed Property

```
property Speed: Double
```

Available In: Client Applications

Specifies the velocity, in meters per second.

> **Note**
> This property is only valid when the SpeedAvailable property is True.

TLocation.SpeedAvailable Property

```
property SpeedAvailable: Boolean
```

Available In: Client Applications

Specifies whether speed information was available as part of the returned location information.

> **Note**
> The Speed property is only valid when the SpeedAvailable property is True.

TLocation.Timestamp Property

```
property Timestamp: DateTime
```

Available In: Client Applications

Specifies the timestamp of when the location information was obtained.

10.146 TLocationServices Component

Unit: WebComps

Inherits From TObject

Available In: Client Applications

The TLocationServices object encapsulates the HTML5 geolocation functionality, which allows the application to determine the physical location of the machine or device running the host web browser (with the user's permission). In addition to the latitude and longitude of the machine or device, additional information such as the altitude, heading, and speed may be available, depending upon the type of device being used to host the web browser.

Note
The component library includes one global instance of this class called LocationServices in the WebComps unit that should be used instead of creating a new instance of the class.

Warning
The HTML5 geolocation functionality is only available in secure contexts (https) in many modern browsers, do please keep this in mind when deciding whether to use such functionality in your application.

Properties	Methods	Events
Error	Create	OnLocationError
HighAccuracy	StartTrackingLocation	OnLocationUpdate
Location	StopTrackingLocation	
MaxAge	UpdateLocation	
Timeout		

TLocationServices.Error Property

```
property Error: TLocationError
```

Available In: Client Applications

Specifies the error condition when the current location cannot be obtained.

Note
This property is only valid after the OnLocationError or OnLocationUpdate event handlers have been executed.

TLocationServices.HighAccuracy Property

```
property HighAccuracy: Boolean
```

Available In: Client Applications

Specifies that the next attempt to obtain the current location information via the UpdateLocation or StartTrackingLocation methods should request a high level of accuracy from the machine or device hosting the web browser. If the machine or device is able to do so, it will use the most accurate method at its disposal to provide the location information.

TLocationServices.Location Property

```
property Location: TLocation
```

Available In: Client Applications

Note
This property is only valid after the OnLocationError or OnLocationUpdate event handlers have been executed.

TLocationServices.MaxAge Property

```
property MaxAge: Integer
```

Available In: Client Applications

Specifies that the next attempt to obtain the current location information via the UpdateLocation or StartTrackingLocation methods should request that any returned location information not have been cached for longer than the specified number of milliseconds. If this property is set to 0 (the default value), any returned location information will not be cached information. If this property is set to -1, any returned location information will only be cached information, no matter how old the information is.

TLocationServices.Timeout Property

```
property Timeout: Integer
```

Available In: Client Applications

Specifies that the next attempt to obtain the current location information via the UpdateLocation or StartTrackingLocation methods should request that any returned location information must be returned within a certain number of milliseconds. If this property is set to 0 (the default value), then the location information must be returned immediately or the OnLocationError event handler will be executed. If this property is set to -1, then the OnLocationError event handler will never be executed due to a timeout, and the OnLocationUpdate event handler will not be executed until the location information is available, no matter how long it takes.

TLocationServices.Create Method

```
constructor Create
```

Available In: Client Applications

Use this method to create a new instance of the TLocationServices class.

TLocationServices.StartTrackingLocation Method

```
procedure StartTrackingLocation
```

Available In: Client Applications

Use this method to start tracking the location information for the machine or device using the HighAccuracy, MaxAge, and Timeout properties to control how the location information is obtained.

After the tracking is started, the OnLocationUpdate event handler will be executed any time the location information changes and the Location property will contain the location information. If the location information cannot be obtained for any reason, then the OnLocationError event handler will be executed and the Error property will contain the error condition.

TLocationServices.StopTrackingLocation Method

```
procedure StopTrackingLocation
```

Available In: Client Applications

Use this method to stop tracking the location information for the machine or device.

TLocationServices.UpdateLocation Method

```
procedure UpdateLocation
```

Available In: Client Applications

Use this method to obtain location information for the machine or device using the HighAccuracy, MaxAge, and Timeout properties to control how the location information is obtained.

If the location information is successfully obtained, the OnLocationUpdate event handler will be executed and the Location property will contain the location information. If the location information cannot be obtained for any reason, then the OnLocationError event handler will be executed and the Error property will contain the error condition.

TLocationServices.OnLocationError Event

```
property OnLocationError: TNotifyEvent
```

Available In: Client Applications

This event is triggered when an attempt to obtain the location information via the UpdateLocation or StartTrackingLocation methods fails. Once this event is triggered, the Error property will contain the error condition.

TLocationServices.OnLocationUpdate Event

```
property OnLocationUpdate: TNotifyEvent
```

Available In: Client Applications

This event is triggered when an attempt to obtain the location information via the UpdateLocation or StartTrackingLocation methods is successful. Once this event is triggered, the Location property will contain the location information.

10.147 TMailer Component

Unit: WebMail

Inherits From TComponent

Available In: Server Applications

The TMailer component is used to send an email via an outbound mail server that supports the SMTP and ESMTP protocols. It supports sending emails using in-the-clear connections (port 25), in-the-clear connections that can be upgraded to TLS connections (port 587), or always-on TLS connections (port 465).

You can specify the email message using either the MailMessage (text content) or MultipartMailMessage (multi-part content) properties.

If text content is specified using the MailMessage property, then the **Content-Type** mail header will automatically be set to "text/plain; charset=utf-8" and the **Content-Transfer-Encoding** mail header will automatically be set to "quoted-printable" in the sent email(s).

If multi-part content is specified, then the **Content-Type** mail header will automatically be set to "multipart/mixed".

> **Note**
> The **Content-Length** mail header is always set automatically by the TMailer component.

Properties	Methods	Events
ClientHost	Execute	OnComplete
Host		OnStart
MailBCC		
MailCC		
MailFrom		
MailMessage		
MailReplyTo		
MailSubject		
MailTo		
MultipartMailMessage		
Password		
Port		
Security		
StatusCode		
StatusText		
Timeout		
Transcript		
UserName		

TMailer.ClientHost Property

```
property ClientHost: String
```

Available In: Server Applications

Specifies the host name to use for identifying the client when connecting to the SMTP server. The default value is "localhost".

TMailer.Host Property

```
property Host: String
```

Available In: Server Applications

Specifies the host name of the SMTP server to use for sending the email(s).

TMailer.MailBCC Property

```
property MailBCC: TStrings
```

Available In: Server Applications

Specifies the email addresses of all tertiary recipients.

TMailer.MailCC Property

```
property MailCC: TStrings
```

Available In: Server Applications

Specifies the email addresses of all secondary recipients.

TMailer.MailFrom Property

```
property MailFrom: String
```

Available In: Server Applications

Specifies the email address of the sender.

TMailer.MailMessage Property

```
property MailMessage: String
```

Available In: Server Applications

Specifies the plain text message to send. This property and the MultipartMailMessage property are mutually-exclusive. Assigning a value to this property will cause all multi-part content to be deleted.

Note
When this property is assigned a value, the **Content-Type** mail header will automatically be set to "text/plain; charset=utf-8" and the **Content-Transfer-Encoding** mail header will automatically be set to "quoted-printable" in the sent email(s).

TMailer.MailReplyTo Property

```
property MailReplyTo: String
```

Available In: Server Applications

Specifies the email address to use for any replies.

TMailer.MailSubject Property

```
property MailSubject: String
```

Available In: Server Applications

Specifies the subject of the message.

TMailer.MailTo Property

```
property MailTo: TStrings
```

Available In: Server Applications

Specifies the email addresses of all primary recipients.

TMailer.MultipartMailMessage Property

```
property MultipartMailMessage: TMIMEContent
```

Available In: Server Applications

Specifies the multi-part MIME message to send. This property and the MailMessage property are mutually-exclusive, and this property supersedes any value assigned to the MailMessage property.

> **Note**
> When at least one MIME part has been defined, the **Content-Type** mail header will automatically be set to "multipart/mixed" in the sent email(s).

TMailer.Password Property

```
property Password: String
```

Available In: Server Applications

Specifies the password to use for authentication on the SMTP server.

TMailer.Port Property

```
property Port: Integer
```

Available In: Server Applications

Specifies the port of the SMTP server to connect to when sending the email(s). The default value is 587.

TMailer.Security Property

```
property Security: TMailSecurity
```

Available In: Server Applications

Specifies the required level of security to use when sending the email(s). The default value is msUpgradeToSecure, which provides the most compatibility with common SMTP servers while making sure that all mail transactions are performed securely.

TMailer.StatusCode Property

```
property StatusCode: Integer
```

Available In: Server Applications

Indicates the SMTP status code returned when the email(s) were attempted to be sent to the SMTP server.

TMailer.StatusText Property

```
property StatusText: String
```

Available In: Server Applications

Indicates the SMTP status text returned when the email(s) were attempted to be sent to the SMTP server.

TMailer.Timeout Property

```
property Timeout: Integer
```

Available In: Server Applications

Specifies the timeout for the request, in seconds, with 0 meaning no timeout.

TMailer.Transcript Property

```
property Transcript: TStrings
```

Available In: Server Applications

Contains the transcript of the SMTP mail session when the email(s) were attempted to be sent to the SMTP server.

TMailer.UserName Property

```
property UserName: String
```

Available In: Server Applications

Specifies the user name to use for authentication on the SMTP server.

TMailer.Execute Method

```
function Execute: Boolean
```

Available In: Server Applications

Use this method to begin the process of sending the email(s) to the SMTP server.

TMailer.OnComplete Event

```
property OnComplete: TMailerEvent
```

Available In: Server Applications

This event is triggered when the email send operation is complete. Use the StatusCode property to determine the status code returned by the mail server and, subsequently, whether the send operation was successful or not.

TMailer.OnStart Event

```
property OnStart: TMailerEvent
```

Available In: Server Applications

This event is triggered when the email send operation is started. A send operation is started when the Execute method is called.

10.148 TMap Component

Unit: WebMaps

Inherits From TMapControl

Available In: Visual Client Applications

The TMap component represents a map control that loads and uses the Google Maps API for map display and geocoding.

Properties	Methods	Events
APIKey		OnAnimationComplete
APIKeyRequired		OnAnimationsComplete
Background		OnAPIError
Border		OnAPILoad
Corners		OnHide
Cursor		OnMove
InsetShadow		OnShow
Locations		OnSize
Options		
OutsetShadow		
Padding		

TMap.APIKey Property

```
property APIKey: String
```

Available In: Visual Client Applications

Google has changed the requirements for using the Google Maps API:

Google Maps Standard Plan Updates

and the TMap control now includes two new properties to accomodate the new API key requirements: APIKeyRequired and APIKey. If you were using the Google Maps API successfully with your application prior to the new requirements, then your IP address will be grandfathered in and you can set the APIKeyRequired property to False and leave the APIKey property blank.

TMap.APIKeyRequired Property

```
property APIKeyRequired: Boolean
```

Available In: Visual Client Applications

Google has changed the requirements for using the Google Maps API:

Google Maps Standard Plan Updates

and the TMap control now includes two new properties to accomodate the new API key requirements: APIKeyRequired and APIKey. If you were using the Google Maps API successfully with your application prior to the new requirements, then your IP address will be grandfathered in and you can set the APIKeyRequired property to False and leave the APIKey property blank.

TMap.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

TMap.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TMap.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TMap.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TMap.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Available In: Visual Client Applications

Specifies the inset shadow for the control.

TMap.Locations Property

```
property Locations: TMapLocations
```

Available In: Visual Client Applications

Provides access to the defined locations for the map.

TMap.Options Property

```
property Options: TMapOptions
```

Available In: Visual Client Applications

Specifies the map options.

TMap.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Available In: Visual Client Applications

Specifies the outset shadow for the control.

TMap.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

TMap.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TMap.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TMap.OnAPIError Event

```
property OnAPIError: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the Google Maps API cannot be loaded due to an error condition.

TMap.OnAPILoad Event

```
property OnAPILoad: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the Google Maps API has been completely loaded and is ready for use.

Note
The Google Maps API is only loaded once and used with every TMap instance, but this event will be triggered for each TMap instance even if the API has already been loaded.

TMap.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TMap.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TMap.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TMap.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

10.149 TMapControl Component

Unit: WebMaps

Inherits From TControl

Available In: Visual Client Applications

The TMapControl control is the base class for map controls that dynamically load and use the Google Maps API. It contains all of the map control functionality in the form of public methods and protected properties/events that descendant classes can use to create customized map controls.

Properties	Methods	Events

10.150 TMapLocation Component

Unit: WebMaps

Inherits From TCollectionItem

Available In: Visual Client Applications

The TMapLocation class represents a named location in a TMap control and contains functionality for getting and setting an address or latitude/longitude, as well as placing markers on the map or changing the map so that the location is at the center.

Properties	Methods	Events
Address		OnClick
Center		
Icon		
Latitude		
Longitude		
ShowMarker		
Title		

TMapLocation.Address Property

```
property Address: String
```

Available In: Visual Client Applications

Specifies an address for the named location. When this property is changed, the Latitude and Longitude properties are set to 0 and the address is geocoded by Google Maps. If the geocoding is successful, then:

- If the Center property is set to True, the location will be centered on the map.
- If the SetMarker property is set to True, the a marker will be placed on the location on the map, and the Title will be displayed as a hint when the mouse hovers over the marker.

TMapLocation.Center Property

```
property Center: Boolean
```

Available In: Visual Client Applications

Specifies whether the location should be set as the center of the map. The default value is False.

Note
Only one named location on the map can have its Center property set to True, and the TMap control will enforce this rule by setting the Center property to False for any other defined locations.

TMapLocation.Icon Property

```
property Icon: String
```

Available In: Visual Client Applications

Specifies the URL for an icon to use in place of the standard location marker.

TMapLocation.Latitude Property

```
property Latitude: Double
```

Available In: Visual Client Applications

Specifies the latitude for the named location. When this property is changed, the Address property is set to ''. If both the Latitude and Longitude properties are assigned non-zero values, then the latitude and longitude are assigned to the map, and:

- If the Center property is set to True, the location will be centered on the map.
- If the SetMarker property is set to True, the a marker will be placed on the location on the map, and the Title will be displayed as a hint when the mouse hovers over the marker.

TMapLocation.Longitude Property

```
property Longitude: Double
```

Available In: Visual Client Applications

Specifies the longitude for the named location. When this property is changed, the Address property is set to ''. If both the Latitude and Longitude properties are assigned non-zero values, then the latitude and longitude are assigned to the map, and:

- If the Center property is set to True, the location will be centered on the map.
- If the SetMarker property is set to True, the a marker will be placed on the location on the map, and the Title will be displayed as a hint when the mouse hovers over the marker.

TMapLocation.ShowMarker Property

```
property ShowMarker: Boolean
```

Available In: Visual Client Applications

Specifies whether a marker should be set for the named location on the map. The default value is False.

TMapLocation.Title Property

```
property Title: String
```

Available In: Visual Client Applications

Specifies the title for the named location to show when the mouse hovers over the marker. This property is only valid when the SetMarker property is set to True.

TMapLocation.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

10.151 TMapLocations Component

Unit: WebMaps

Inherits From TCollection

Available In: Visual Client Applications

The TMapLocations class represents the list of named locations in a TMap control and contains functionality for managing the locations.

Properties	Methods	Events
Location		

TMapLocations.Location Property

```
property Location[Index: Integer]: TMapLocation

property Location[const Name: String]: TMapLocation
```

Available In: Visual Client Applications

Accesses all locations defined for the map by index or by name.

10.152 TMapOption Component

Unit: WebMaps

Inherits From TPersistent

Available In: Visual Client Applications

The TMapOption class is the base class for any complex TMap control mapping options.

Properties	Methods	Events
	Create	

TMapOption.Create Method

```
constructor Create(AMap: TMapControl; AParent: TMapOption)
```

Available In: Visual Client Applications

Use this method to create a new instance of the TMapOption class. The AMap parameter indicates the map control instance that will manage the option, and the AParent parameter indicates the parent option, if any, that the option is contained within. The parent option is used to aggregate change management at the outermost option so as to avoid excessively triggering change notifications in the map control.

10.153 TMapOptions Component

Unit: WebMaps

Inherits From TMapOption

Available In: Visual Client Applications

The TMapOptions class represents a list of map options for a TMap control. These map options correspond to the map options available for maps in the Google Maps API.

Properties	Methods	Events
DisableDblClickZoom		
DisableDefaultUI		
Draggable		
Heading		
KeyboardShortCuts		
MapType		
MapTypeControl		
MapTypeControlOptions		
MaxZoom		
MinZoom		
OverviewMapControl		
OverviewMapControlOptions		
PanControl		
PanControlOptions		
RotateControl		
RotateControlOptions		
ScaleControl		
ScrollWheel		
StreetViewControl		
StreetViewControlOptions		
Tilt		
Zoom		
ZoomControl		
ZoomControlOptions		

TMapOptions.DisableDblClickZoom Property

```
property DisableDblClickZoom: Boolean
```

Available In: Visual Client Applications

Enables/disables zoom and center on double click. Enabled by default.

TMapOptions.DisableDefaultUI Property

```
property DisableDefaultUI: Boolean
```

Available In: Visual Client Applications

Enables/disables all default UI. May be overridden individually.

TMapOptions.Draggable Property

```
property Draggable: Boolean
```

Available In: Visual Client Applications

If False, prevents the map from being dragged. Dragging is enabled by default.

TMapOptions.Heading Property

```
property Heading: Double
```

Available In: Visual Client Applications

The heading for aerial imagery in degrees measured clockwise from cardinal direction North. Headings are snapped to the nearest available angle for which imagery is available.

TMapOptions.KeyboardShortCuts Property

```
property KeyboardShortCuts: Boolean
```

Available In: Visual Client Applications

If false, prevents the map from being controlled by the keyboard. Keyboard shortcuts are enabled by default.

TMapOptions.MapType Property

```
property MapType: TMapType
```

Available In: Visual Client Applications

Specifies the initial map type. The default value is mtRoadmap.

TMapOptions.MapTypeControl Property

```
property MapTypeControl: Boolean
```

Available In: Visual Client Applications

The initial enabled/disabled state of the map type control.

TMapOptions.MapTypeControlOptions Property

```
property MapTypeControlOptions: TMapTypeControlOptions
```

Available In: Visual Client Applications

The initial display options for the map type control.

TMapOptions.MaxZoom Property

```
property MaxZoom: Integer
```

Available In: Visual Client Applications

The maximum zoom level which will be displayed on the map.

TMapOptions.MinZoom Property

```
property MinZoom: Integer
```

Available In: Visual Client Applications

The minimum zoom level which will be displayed on the map.

TMapOptions.OverviewMapControl Property

```
property OverviewMapControl: Boolean
```

Available In: Visual Client Applications

The enabled/disabled state of the overview map control.

TMapOptions.OverviewMapControlOptions Property

```
property OverviewMapControlOptions: TOverviewMapControlOptions
```

Available In: Visual Client Applications

The display options for the overview map control.

TMapOptions.PanControl Property

```
property PanControl: Boolean
```

Available In: Visual Client Applications

The enabled/disabled state of the pan control.

TMapOptions.PanControlOptions Property

```
property PanControlOptions: TPanControlOptions
```

Available In: Visual Client Applications

The display options for the pan control.

TMapOptions.RotateControl Property

```
property RotateControl: Boolean
```

Available In: Visual Client Applications

The enabled/disabled state of the rotate control.

TMapOptions.RotateControlOptions Property

```
property RotateControlOptions: TRotateControlOptions
```

Available In: Visual Client Applications

The display options for the rotate control.

TMapOptions.ScaleControl Property

```
property ScaleControl: Boolean
```

Available In: Visual Client Applications

The initial enabled/disabled state of the scale control.

TMapOptions.ScrollWheel Property

```
property ScrollWheel: Boolean
```

Available In: Visual Client Applications

If false, disables mouse scrollwheel zooming on the map. The mouse scrollwheel is enabled by default.

TMapOptions.StreetViewControl Property

```
property StreetViewControl: Boolean
```

Available In: Visual Client Applications

The initial enabled/disabled state of the street view pegman control. This control is part of the default UI, and should be set to false when displaying a map type on which the street view road overlay should not appear (e.g. a non-Earth map type).

TMapOptions.StreetViewControlOptions Property

```
property StreetViewControlOptions: TStreetViewControlOptions
```

Available In: Visual Client Applications

The initial display options for the street view pegman control.

TMapOptions.Tilt Property

```
property Tilt: TMapTilt
```

Available In: Visual Client Applications

Controls the automatic switching behavior for the angle of incidence of the map. The only allowed values are 0 and 45. The value 0 causes the map to always use a 0° overhead view regardless of the zoom level and viewport. The value 45 causes the tilt angle to automatically switch to 45 whenever 45° imagery is available for the current zoom level and viewport, and switch back to 0 whenever 45° imagery is not available (this is the default behavior). 45° imagery is only available for the mtSatellite and mtHybrid map types, within some locations, and at some zoom levels.

TMapOptions.Zoom Property

```
property Zoom: Integer
```

Available In: Visual Client Applications

The initial map zoom level.

TMapOptions.ZoomControl Property

```
property ZoomControl: Boolean
```

Available In: Visual Client Applications

The enabled/disabled state of the zoom control.

TMapOptions.ZoomControlOptions Property

```
property ZoomControlOptions: TZoomControlOptions
```

Available In: Visual Client Applications

The display options for the zoom control.

10.154 TMapTypeControlMapTypes Component

Unit: WebMaps

Inherits From TMapOption

Available In: Visual Client Applications

The TMapTypeControlMapTypes class represents the list of map types that should be available for user selection using the map type control in a TMap control. These map types correspond to the map types available for maps in the Google Maps API.

Properties	Methods	Events
Hybrid		
Roadmap		
Satellite		
Terrain		

TMapTypeControlMapTypes.Hybrid Property

```
property Hybrid: Boolean
```

Available In: Visual Client Applications

Include hybrid as a selectable map type.

TMapTypeControlMapTypes.Roadmap Property

```
property Roadmap: Boolean
```

Available In: Visual Client Applications

Include roadmap as a selectable map type.

TMapTypeControlMapTypes.Satellite Property

```
property Satellite: Boolean
```

Available In: Visual Client Applications

Include satellite as a selectable map type.

TMapTypeControlMapTypes.Terrain Property

```
property Terrain: Boolean
```

Available In: Visual Client Applications

Include terrain as a selectable map type.

10.155 TMapTypeControlOptions Component

Unit: WebMaps

Inherits From TMapOption

Available In: Visual Client Applications

The TMapTypeControlOptions class controls how the map type control is configured in a TMap control. These map type control options correspond to the map type control options available for maps in the Google Maps API.

Properties	Methods	Events
MapTypes		
Position		
Style		

TMapTypeControlOptions.MapTypes Property

```
property MapTypes: TMapTypeControlMapTypes
```

Available In: Visual Client Applications

Specifies which map types to show in the map type control.

TMapTypeControlOptions.Position Property

```
property Position: TMapControlPosition
```

Available In: Visual Client Applications

Specifies the position of the map type control.

TMapTypeControlOptions.Style Property

```
property Style: TMapTypeControlStyle
```

Available In: Visual Client Applications

Specifies the style of the map type control.

10.156 TMargins Component

Unit: WebUI

Inherits From TBoundingAttribute

Available In: Visual Client Applications

The TMargins class represents the margins of a UI element or control. The margins affect how a UI element or control is positioned and sized within the current layout rectangle when using the layout functionality. Please see the Layout Management topic for more information on how margins are used with the layout functionality.

Properties	Methods	Events

10.157 TMediaControl Component

Unit: WebMedia

Inherits From TBindableColumnControl

Available In: Visual Client Applications

The TMediaControl control is the base class for media controls, and contains all of the core media functionality in the form of public methods and protected properties/events that descendant classes can use to create customized media controls.

Properties	Methods	Events
	CanPlayMedia	
	Pause	
	Play	

TMediaControl.CanPlayMedia Method

```
function CanPlayMedia(const MIMEType: String): TCanPlayMedia
```

Available In: Visual Client Applications

Call this method to determine if the media control can play a specific type of media.

The MIMEType parameter indicates the MIME type of the media that you wish to test for playback capabilities.

TMediaControl.Pause Method

```
procedure Pause
```

Available In: Visual Client Applications

Call this method to pause playback of the media.

TMediaControl.Play Method

```
procedure Play
```

Available In: Visual Client Applications

Call this method to start or resume playback of the media.

10.158 TMediaElement Component

Unit: WebUI

Inherits From TElement

Available In: Visual Client Applications

The TMediaElement class is the element class for media UI elements, and contains all of the base media playback functionality that is used by the TAudioElement and TVideoElement.

> **Note**
> This element does not provide support for media playback at design-time, and the applicable playback methods and properties are all stubs. Also, this element is never instantiated directly. Only the TAudioElement and TVideoElement UI elements are instantiated.

Properties	Methods	Events
AutoPlay	CanPlayMedia	
CurrentSource	Pause	
CurrentTime	Play	
DefaultPlaybackRate		
Duration		
Ended		
Loop		
Muted		
NetworkState		
Paused		
PlaybackRate		
Preload		
ReadyState		
Seeking		
ShowControls		
Source		
Volume		

TMediaElement.AutoPlay Property

```
property AutoPlay: Boolean
```

Available In: Visual Client Applications

Specifies that the media should begin playing as soon as enough data has been loaded to allow playback. The default value is False.

TMediaElement.CurrentSource Property

```
property CurrentSource: String
```

Available In: Visual Client Applications

Indicates the URL of the current media being loaded and/or played.

TMediaElement.CurrentTime Property

```
property CurrentTime: Double
```

Available In: Visual Client Applications

Indicates the current playback time, in seconds. Setting this property to a new value will cause the media to skip to the specified time.

TMediaElement.DefaultPlaybackRate Property

```
property DefaultPlaybackRate: Double
```

Available In: Visual Client Applications

Specifies the default playback rate, with 1 being normal playback, less than 1 being slower playback, and greater than 1 being faster playback. The default value is 1.

Note
The volume will normally be automatically muted when playing media faster or slower than the normal playback rate.

TMediaElement.Duration Property

```
property Duration: Double
```

Available In: Visual Client Applications

Indicates the length of the media in seconds. If the duration has not been determined, this property will return 0.

TMediaElement.Ended Property

```
property Ended: Boolean
```

Available In: Visual Client Applications

Indicates that the end of the media has been reached.

TMediaElement.Loop Property

```
property Loop: Boolean
```

Available In: Visual Client Applications

Specifies that the media playback should automatically restart at the beginning once the end has been reached. The default value is False.

TMediaElement.Muted Property

```
property Muted: Boolean
```

Available In: Visual Client Applications

Specifies that the playback volume should be muted. The default valuse is False.

TMediaElement.NetworkState Property

```
property NetworkState: TMediaNetworkState
```

Available In: Visual Client Applications

Indicates the network state of the media loading/playback.

TMediaElement.Paused Property

```
property Paused: Boolean
```

Available In: Visual Client Applications

Indicates that media playback is paused, either by the user pausing the media via the user interface when the ShowControls property is True, or by the application calling the Pause method. The default value is False.

TMediaElement.PlaybackRate Property

```
property PlaybackRate: Double
```

Available In: Visual Client Applications

Specifies the playback rate, with 1 being normal playback, less than 1 being slower playback, and greater than 1 being faster playback. The default value is 1.

Note
The volume will normally be automatically muted when playing media faster or slower than the normal playback rate.

TMediaElement.Preload Property

```
property Preload: TMediaPreload
```

Available In: Visual Client Applications

Specifies how much of the current media data should be loaded before playback begins.

TMediaElement.ReadyState Property

```
property ReadyState: TMediaReadyState
```

Available In: Visual Client Applications

Indicates whether the media is ready for playback, and if so, a general description of what media data has been loaded.

TMediaElement.Seeking Property

```
property Seeking: Boolean
```

Available In: Visual Client Applications

Indicates that media is switching to a new playback location, either by the user changing the playback location in the media via the user interface when the ShowControls property is True, or by the application setting the CurrentTime property.

TMediaElement.ShowControls Property

```
property ShowControls: Boolean
```

Available In: Visual Client Applications

Specifies whether the element should show the native user interface for the media being played. The TAudioElement UI element instances will show an audio player interface, and the TVideoElement UI element instances will show a video player interface.

TMediaElement.Source Property

```
property Source: String
```

Available In: Visual Client Applications

Specifies the URL of the media to be loaded into the media element. Whenever this property is changed, the existing media is cleared and the new media will start downloading from the web server. Please review the events available for this element in order to get more information on detecting and handling the loading/playback of the media.

TMediaElement.Volume Property

```
property Volume: Integer
```

Available In: Visual Client Applications

Specifies the playback volume of the audio for the media. The volume can be set between 0 and 100.

TMediaElement.CanPlayMedia Method

```
function CanPlayMedia(const MIMEType: String): TCanPlayMedia
```

Available In: Visual Client Applications

Call this method to determine if the media element can play a specific type of media. The MIMEType parameter indicates the MIME type of the media that you wish to test for playback capabilities.

TMediaElement.Pause Method

```
procedure Pause
```

Available In: Visual Client Applications

Call this method to pause playback of the media.

TMediaElement.Play Method

```
procedure Play
```

Available In: Visual Client Applications

Call this method to start or resume playback of the media.

10.159 TMemoryStream Component

Unit: WebSrvr

Inherits From TStream

Available In: Server Applications

The TMemoryStream class is used to perform read/write operations on a dynamically-sized block of memory using the ancestor TStream class properties/methods.

Properties	Methods	Events
	Create	

TMemoryStream.Create Method

```
constructor Create(BlockSize: Integer=DEFAULT_BLOCK_SIZE)
```

Available In: Server Applications

Use this method to create a new instance of the TMemoryStream class using the specified block size. The block size determines the unit used for allocating/resizing the memory for the stream. For example, if the memory stream is created with a block size of 8192, then 8192 bytes of memory will be allocated for the memory stream when an initial write of an integer (4 bytes) using the TStream WriteInt32 method.

10.160 TMenu Component

Unit: WebMenus

Inherits From TMenuControl

Available In: Visual Client Applications

The TMenu component represents a vertical menu control that can be used for side menus or popup menus.

Properties	Methods	Events
Background	NewItem	OnAnimationComplete
Border	NewSeparatorItem	OnAnimationsComplete
Corners	Popup	OnEnter
Cursor		OnExit
Hint		OnHide
Opacity		OnItemClick
OutsetShadow		OnMove
TabOrder		OnShow
TabStop		OnSize

TMenu.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

TMenu.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TMenu.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TMenu.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TMenu.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TMenu.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TMenu.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Available In: Visual Client Applications

Specifies the outset shadow for the control.

TMenu.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TMenu.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TMenu.NewItem Method

```
function NewItem: TMenuItem
```

Available In: Visual Client Applications

Use this method to create a new TMenuItem instance and append it to the current TMenu instance as the last menu item.

TMenu.NewSeparatorItem Method

```
function NewSeparatorItem: TMenuSeparatorItem
```

Available In: Visual Client Applications

Use this method to create a new TMenuSeparatorItem instance and append it to the current TMenu instance as the last menu separator item.

TMenu.Popup Method

```
procedure Popup(X,Y: Integer)
```

Available In: Visual Client Applications

Use this method to display the menu as a popup menu. When a menu is displayed as a popup, then it will behave as a popup and automatically be hidden whenever the menu loses focus.

TMenu.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TMenu.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TMenu.OnEnter Event

```
property OnEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control obtains input focus.

TMenu.OnExit Event

```
property OnExit: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control loses input focus.

TMenu.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TMenu.OnItemClick Event

```
property OnItemClick: TClickEvent
```

Available In: Visual Client Applications

This event is triggered whenever a menu item is clicked.

Return True to allow the default click behavior and False to prevent the default click behavior from occurring.

TMenu.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TMenu.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TMenu.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

10.161 TMenuBar Component

Unit: WebMenus

Inherits From TMenuControl

Available In: Visual Client Applications

The TMenu component represents a horizontal menu control that can be used for main menus.

Properties	Methods	Events
Background	NewItem	OnAnimationComplete
Border	NewSeparatorItem	OnAnimationsComplete
Corners		OnEnter
Cursor		OnExit
Hint		OnHide
Opacity		OnItemClick
OutsetShadow		OnMove
TabOrder		OnShow
TabStop		OnSize

TMenuBar.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

TMenuBar.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TMenuBar.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TMenuBar.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TMenuBar.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TMenuBar.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TMenuBar.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Available In: Visual Client Applications

Specifies the outset shadow for the control.

TMenuBar.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TMenuBar.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TMenuBar.NewItem Method

```
function NewItem: TMenuBarItem
```

Available In: Visual Client Applications

Use this method to create a new TMenuBarItem instance and append it to the current TMenuBar instance as the last menu bar item.

TMenuBar.NewSeparatorItem Method

```
function NewSeparatorItem: TMenuBarSeparatorItem
```

Available In: Visual Client Applications

Use this method to create a new TMenuBarSeparatorItem instance and append it to the current TMenuBar instance as the last menu bar separator item.

TMenuBar.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TMenuBar.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TMenuBar.OnEnter Event

```
property OnEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control obtains focus.

TMenuBar.OnExit Event

```
property OnExit: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control loses focus.

TMenuBar.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TMenuBar.OnItemClick Event

```
property OnItemClick: TClickEvent
```

Available In: Visual Client Applications

This event is triggered whenever a menu bar item is clicked.

Return True to allow the default click behavior and False to prevent the default click behavior from occurring.

TMenuBar.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TMenuBar.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TMenuBar.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

10.162 TMenuBarItem Component

Unit: WebMenus

Inherits From TMenuItemControl

Available In: Visual Client Applications

The TMenuBarItem component represents a textual menu item within a TMenuBar control.

Properties	Methods	Events
AutoWidth		OnCaptureEnd
Caption		OnCaptureStart
Cursor		OnCapturing
Enabled		OnClick
Font		OnContextMenu
Hint		OnEnter
Icon		OnExit
SubMenu		OnHide
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnShow
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TMenuBarItem.AutoWidth Property

```
property AutoWidth: Boolean
```

Available In: Visual Client Applications

Specifies whether the width of the menu bar item should be automatically set based upon the Caption, Icon, and Font properties.

TMenuBarItem.Caption Property

```
property Caption: String
```

Available In: Visual Client Applications

Specifies the textual caption to display in the control. The default value is ''.

TMenuBarItem.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TMenuBarItem.Enabled Property

```
property Enabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TMenuBarItem.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TMenuBarItem.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TMenuBarItem.Icon Property

```
property Icon: TIconProperties
```

Available In: Visual Client Applications

Specifies the properties of the icon used with the control.

TMenuBarItem.SubMenu Property

```
property SubMenu: TMenu
```

Available In: Visual Client Applications

Specifies the sub-menu for this menu bar item. Setting this property to an instance of the TMenu control allows the menu bar item to popup the sub-menu when the menu bar item is clicked.

TMenuBarItem.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TMenuBarItem.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TMenuBarItem.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TMenuBarItem.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TMenuBarItem.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TMenuBarItem.OnEnter Event

```
property OnEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control obtains focus.

TMenuBarItem.OnExit Event

```
property OnExit: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control loses focus.

TMenuBarItem.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TMenuBarItem.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TMenuBarItem.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TMenuBarItem.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TMenuBarItem.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TMenuBarItem.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TMenuBarItem.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TMenuBarItem.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TMenuBarItem.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TMenuBarItem.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TMenuBarItem.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.163 TMenuBarSeparatorItem Component

Unit: WebMenus

Inherits From TMenuItemControl

Available In: Visual Client Applications

The TMenuBarSeparatorItem component represents a menu item separator within a TMenuBar control.

Properties	Methods	Events
		OnHide
		OnShow

TMenuBarSeparatorItem.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TMenuBarSeparatorItem.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

10.164 TMenuControl Component

Unit: WebMenus

Inherits From TControl

Available In: Visual Client Applications

The TMenuControl control is the base class for vertical menu controls, and contains all of the core vertical menu functionality in the form of public methods and protected properties/events that descendant classes can use to create customized menu controls.

Properties	Methods	Events
ItemCount	FirstItem	
ItemIndex	LastItem	
Items	MakeItemVisible	
VisibleItemCount	NextItem	
VisibleItems	PriorItem	

TMenuControl.ItemCount Property

```
property ItemCount: Integer
```

Available In: Visual Client Applications

Indicates the number of menu items in the menu control.

TMenuControl.ItemIndex Property

```
property ItemIndex: Integer
```

Available In: Visual Client Applications

Indicates the index of the currently-selected menu item in the menu control, or -1 if no menu item is selected.

TMenuControl.Items Property

```
property Items[AIndex: Integer]: TMenuItemControl
```

Available In: Visual Client Applications

Accesses the menu items in the menu control by index.

TMenuControl.VisibleItemCount Property

```
property VisibleItemCount: Integer
```

Available In: Visual Client Applications

Indicates the number of visible menu items in the menu control.

TMenuControl.VisibleItems Property

```
property VisibleItems[AIndex: Integer]: TMenuItemControl
```

Available In: Visual Client Applications

Accesses the visible menu items in the menu control by index.

TMenuControl.FirstItem Method

```
procedure FirstItem
```

Available In: Visual Client Applications

Use this method to move focus to the first menu item in the menu control.

TMenuControl.LastItem Method

```
procedure LastItem
```

Available In: Visual Client Applications

Use this method to move focus to the last menu item in the menu control.

TMenuControl.MakeItemVisible Method

```
procedure MakeItemVisible(AItem: TMenuItemControl)
```

Available In: Visual Client Applications

Use this method to ensure that the specified menu item is visible.

TMenuControl.NextItem Method

```
procedure NextItem
```

Available In: Visual Client Applications

Use this method to move focus to the next menu item, if one exists, in the menu control.

TMenuControl.PriorItem Method

```
procedure PriorItem
```

Available In: Visual Client Applications

Use this method to move focus to the prior menu item, if one exists, in the menu control.

10.165 TMenuItem Component

Unit: WebMenus

Inherits From TMenuItemControl

Available In: Visual Client Applications

The TMenuItem component represents a textual menu item within a TMenu control.

Properties	Methods	Events
AutoHeight		OnCaptureEnd
Caption		OnCaptureStart
Cursor		OnCapturing
Enabled		OnClick
Font		OnContextMenu
Hint		OnEnter
Icon		OnExit
SubMenu		OnHide
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnShow
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TMenuItem.AutoHeight Property

```
property AutoHeight: Boolean
```

Available In: Visual Client Applications

Specifies whether the height of the menu item should be automatically set based upon the Caption and Font properties.

TMenuItem.Caption Property

```
property Caption: String
```

Available In: Visual Client Applications

Specifies the textual caption to display in the control. The default value is ''.

TMenuItem.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TMenuItem.Enabled Property

```
property Enabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TMenuItem.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TMenuItem.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TMenuItem.Icon Property

```
property Icon: TIconProperties
```

Available In: Visual Client Applications

Specifies the properties of the icon used with the control.

TMenuItem.SubMenu Property

```
property SubMenu: TMenu
```

Available In: Visual Client Applications

Specifies the sub-menu for this menu item. Setting this property to an instance of the TMenu control allows the menu item to popup the sub-menu when the menu item is clicked.

TMenuItem.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TMenuItem.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TMenuItem.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TMenuItem.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TMenuItem.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TMenuItem.OnEnter Event

```
property OnEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control obtains input focus.

TMenuItem.OnExit Event

```
property OnExit: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control loses input focus.

TMenuItem.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TMenuItem.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TMenuItem.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TMenuItem.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TMenuItem.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TMenuItem.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TMenuItem.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TMenuItem.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TMenuItem.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TMenuItem.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TMenuItem.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.166 TMenuItemControl Component

Unit: WebMenus

Inherits From TControl

Available In: Visual Client Applications

The TMenuItemControl control is the base class for menu control items, including normal textual menu items and menu item separators, and contains all of the core vertical menu item functionality in the form of public methods and protected properties/events that descendant classes can use to create customized menu item controls.

Properties	Methods	Events
Index	HideSubMenu	
ParentMenu	ShowSubMenu	

TMenuItemControl.Index Property

```
property Index: Integer
```

Available In: Visual Client Applications

Indicates the position of the menu item in the parent menu.

TMenuItemControl.ParentMenu Property

```
property ParentMenu: TMenuControl
```

Available In: Visual Client Applications

Indicates the parent menu that contains the menu item.

TMenuItemControl.HideSubMenu Method

```
procedure HideSubMenu
```

Available In: Visual Client Applications

Use this method to hide the sub-menu, if one is set for the menu item and is currently visible.

TMenuItemControl.ShowSubMenu Method

```
procedure ShowSubMenu
```

Available In: Visual Client Applications

Use this method to show the sub-menu, if one is set for the menu item and is currently not visible.

10.167 TMenuSeparatorItem Component

Unit: WebMenus

Inherits From TMenuItemControl

Available In: Visual Client Applications

The TMenuSeparatorItem component represents a menu item separator within a TMenu control.

Properties	Methods	Events
		OnHide
		OnShow

TMenuSeparatorItem.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TMenuSeparatorItem.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

10.168 TMessageDialog Component

Unit: WebForms

Inherits From TDialogControl

Available In: Visual Client Applications

The TMessageDialog component represents a message dialog control. Please see the Showing Message Dialogs for more information on using message dialogs.

Properties	Methods	Events
AllowClose	AddButton	OnAnimationComplete
AllowMove		OnAnimationsComplete
ButtonCount		OnCaptureEnd
Buttons		OnCaptureStart
Caption		OnCapturing
CloseOnEscape		OnClick
Corners		OnClose
Cursor		OnCloseQuery
DialogType		OnContextMenu
Message		OnDblClick
Opacity		OnHide
OutsetShadow		OnKeyDown
		OnKeyPress
		OnKeyUp
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMouseWheel
		OnMove
		OnResult
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TMessageDialog.AllowClose Property

```
property AllowClose: Boolean
```

Available In: Visual Client Applications

Specifies whether the close button should be shown in the caption bar of the dialog.

TMessageDialog.AllowMove Property

```
property AllowMove: Boolean
```

Available In: Visual Client Applications

Specifies whether the user can press and hold a mouse or touch on the caption bar and drag the container dialog to a new position.

TMessageDialog.ButtonCount Property

```
property ButtonCount: Integer
```

Available In: Visual Client Applications

Indicates the number of dialog buttons added to the dialog.

TMessageDialog.Buttons Property

```
property Buttons[Index: Integer]: TDialogButton
```

Available In: Visual Client Applications

Accesses the dialog buttons that have been added to the dialog.

TMessageDialog.Caption Property

```
property Caption: String
```

Available In: Visual Client Applications

Specifies the caption to display in the caption bar of the dialog.

TMessageDialog.CloseOnEscape Property

```
property CloseOnEscape: Boolean
```

Available In: Visual Client Applications

Specifies whether the dialog is automatically closed when the user hits the Escape key. The default value is True.

Note
If there is a TDialogButton instance on the dialog with its ModalCancel property set to True, then the button will be clicked when the escape key is pressed and this property will ignored.

TMessageDialog.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TMessageDialog.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TMessageDialog.DialogType Property

```
property DialogType: TMsgDlgType
```

Available In: Visual Client Applications

Specifies the type of dialog. The dialog type determines which icon is displayed in the dialog.

TMessageDialog.Message Property

```
property Message: String
```

Available In: Visual Client Applications

Specifies the message to display in the dialog.

TMessageDialog.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TMessageDialog.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Available In: Visual Client Applications

Specifies the outset shadow for the control.

TMessageDialog.AddButton Method

```
function AddButton(ButtonType: TMsgDlgBtn): TDialogButton
```

Available In: Visual Client Applications

Use this method to add a new dialog button to the dialog. The button type determines the caption and behavior of the dialog button.

TMessageDialog.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TMessageDialog.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TMessageDialog.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TMessageDialog.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TMessageDialog.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TMessageDialog.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TMessageDialog.OnClose Event

```
property OnClose: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the dialog is closed by the user via the caption bar close button, or when the Close method is called.

TMessageDialog.OnCloseQuery Event

```
property OnCloseQuery: TCloseQueryEvent
```

Available In: Visual Client Applications

This event is triggered when the dialog is closed by the user via the caption bar close button, or when the Close method is called.

Return True to allow the close to continue, or False to prevent the dialog from closing.

TMessageDialog.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TMessageDialog.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TMessageDialog.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TMessageDialog.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and the user presses a key or key combination.

TMessageDialog.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and presses/releases a key or key combination.

TMessageDialog.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and the user releases a key or key combination.

TMessageDialog.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TMessageDialog.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TMessageDialog.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TMessageDialog.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TMessageDialog.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TMessageDialog.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

Available In: Visual Client Applications

This event is triggered whenever the mouse wheel is rotated forward or backward.

TMessageDialog.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TMessageDialog.OnResult Event

```
property OnResult: TMsgDlgResultEvent
```

Available In: Visual Client Applications

This event is triggered after a modal dialog has been closed and a modal result is available.

TMessageDialog.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TMessageDialog.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TMessageDialog.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TMessageDialog.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TMessageDialog.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TMessageDialog.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.169 TMIMEContent Component

Unit: WebMIME

Inherits From TObject

Available In: Server Applications

The TMIMEContent class is used by the TMailer component to represent multi-part MIME content in an email.

Properties	Methods	Events
Count	Add	
Part	Clear	
	Create	
	Delete	

TMIMEContent.Count Property

```
property Count: Integer
```

Available In: Server Applications

Indicates the total number of MIME parts.

TMIMEContent.Part Property

```
property Part[Index: Integer]: TMIMEContentPart
```

Available In: Server Applications

Accesses the MIME part at the specified index.

TMIMEContent.Add Method

```
function Add: TMIMEContentPart
```

Available In: Server Applications

Use this method to add a new MIME part.

TMIMEContent.Clear Method

```
procedure Clear
```

Available In: Server Applications

Use this method to delete all MIME parts.

TMIMEContent.Create Method

```
constructor Create
```

Available In: Server Applications

Use this method to create a new instance of the TMIMEContent class.

Note
Because the MIME content is automatically created and included as part of the TMailer component, you will most likely never need to call this method.

TMIMEContent.Delete Method

```
procedure Delete(Index: Integer)
```

Available In: Server Applications

Use this method to delete the MIME part at the specified index.

10.170 TMIMEContentPart Component

Unit: WebMIME

Inherits From TObject

Available In: Server Applications

The TMIMEContentPart class is used by the TMIMEContent class to represent one part of the multi-part MIME content in an email.

Properties	Methods	Events
Content	Create	
ContentStream		
Headers		

TMIMEContentPart.Content Property

```
property Content: String
```

Available In: Server Applications

Specifies the textual content to include as the MIME part. This property and the ContentStream property are mutually-exclusive. If both properties are assigned a value, then the ContentStream property will take precedence.

Note
If this property is assigned a value and the **Content-Type** header is not specified using the Headers property, it will be automatically set to "text/plain; charset=utf-8" when the email(s) are sent. In addition, if the **Content-Transfer-Encoding** header is not specified, it will automatically set to "quoted-printable" when the email(s) are sent.

TMIMEContentPart.ContentStream Property

```
property ContentStream: TStream
```

Available In: Server Applications

Specifies a TStream descendant class instance that contains the binary content to include as the MIME part. This property and the Content property are mutually-exclusive. If both properties are assigned a value, then the ContentStream property will take precedence.

Note
If this property is assigned a value and the **Content-Type** header is not specified using the Headers property, it will be automatically set to "application/octet-stream" when the email(s) are sent. In addition, if the **Content-Transfer-Encoding** header is not specified, it will automatically set to "base64" when the email(s) are sent.

Warning
This is just a reference to the TStream descendant class instance, so please make sure that you do not free this class instance before the TMailer Execute method is called.

TMIMEContentPart.Headers Property

```
property Headers: THeaders
```

Available In: Server Applications

Provides access to the MIME headers for the MIME part.

TMIMEContentPart.Create Method

```
constructor Create
```

Available In: Server Applications

Use this method to create a new instance of the TMIMEContentPart class.

10.171 TModalOverlay Component

Unit: WebForms

Inherits From TControl

Available In: Visual Client Applications

The TModalOverlay component represents the modal overlay area that is used to cover all existing controls when a form is shown modally. This component provides access to the overlay so that its appearance can be customized. Please see the Creating and Showing Forms topic for more information on showing forms.

Properties	Methods	Events
Background		
CloseOnClick		

TModalOverlay.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

TModalOverlay.CloseOnClick Property

```
property CloseOnClick: Boolean
```

Available In: Visual Client Applications

Specifies whether clicking on the modal overlay will automatically close all visible modal forms. The default value is False.

10.172 TMultiLineEdit Component

Unit: WebEdits

Inherits From TMultiLineEditControl

Available In: Visual Client Applications

The TMultiLineEdit component represents a multi-line edit control. A a multi-line edit control allows the user to directly enter an input value using the keyboard, and the input value can contain multiple lines and be word-wrapped.

Properties	Methods	Events
Alignment		OnAnimationComplete
Cursor		OnAnimationsComplete
DataColumn		OnCaptureEnd
DataSet		OnCaptureStart
Direction		OnCapturing
Enabled		OnChange
Font		OnClick
Hint		OnContextMenu
Lines		OnDblClick
MaxLength		OnEnter
ReadOnly		OnExit
ScrollBars		OnHide
ScrollSupport		OnKeyDown
SpellCheck		OnKeyPress
TabOrder		OnKeyUp
TabStop		OnMouseDown
Text		OnMouseEnter
WordWrap		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMouseWheel
		OnMove
		OnScroll
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd

		OnTouchMove
		OnTouchScroll
		OnTouchStart

TMultiLineEdit.Alignment Property

```
property Alignment: TContentAlignment
```

Available In: Visual Client Applications

Specifies the alignment of the input value for the control.

TMultiLineEdit.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TMultiLineEdit.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TMultiLineEdit.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

TMultiLineEdit.Direction Property

```
property Direction: TContentDirection
```

Available In: Visual Client Applications

Specifies the direction in which the text is displayed/edited.

TMultiLineEdit.Enabled Property

```
property Enabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TMultiLineEdit.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TMultiLineEdit.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TMultiLineEdit.Lines Property

```
property Lines: TStrings
```

Available In: Visual Client Applications

Specifies the lines to display and edit in the control.

> **Note**
> Updating this property will automatically update the Text property, and vice-versa.

TMultiLineEdit.MaxLength Property

```
property MaxLength: Integer
```

Available In: Visual Client Applications

Specifies the maximum allowable length, in characters, of the Text property for the control. A value of 0 specifies an unlimited allowable length.

TMultiLineEdit.ReadOnly Property

```
property ReadOnly: Boolean
```

Available In: Visual Client Applications

Specifies whether the control's input value can be modified by the user. The default value is False.

> **Note**
> The input value can always be programmatically modified.

TMultiLineEdit.ScrollBars Property

```
property ScrollBars: TScrollBars
```

Available In: Visual Client Applications

Specifies which scrollbars to show, if any.

Note
Even if this property is set to sbHorizontal, sbVertical, or sbBoth, a scrollbar will only be shown if the size of the contents of the control exceed the client rectangle for the control.

TMultiLineEdit.ScrollSupport Property

```
property ScrollSupport: TScrollSupport
```

Available In: Visual Client Applications

Specifies the directions in which the control can be scrolled, if any.

> **Note**
> This property only applies to scrolling via touch and mouse wheel movements, and does not apply to programmatic scrolling or scrolling via the scroll bars themselves.

TMultiLineEdit.SpellCheck Property

```
property SpellCheck: Boolean
```

Available In: Visual Client Applications

Specifies whether spell-checking will be enabled for the control.

TMultiLineEdit.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TMultiLineEdit.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TMultiLineEdit.Text Property

```
property Text: String
```

Available In: Visual Client Applications

Specifies the control's input value as a string.

> **Note**
> Updating this property will automatically update the Lines property, and vice-versa.

TMultiLineEdit.WordWrap Property

```
property WordWrap: Boolean
```

Available In: Visual Client Applications

Specifies whether the content should be word-wrapped.

TMultiLineEdit.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TMultiLineEdit.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TMultiLineEdit.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TMultiLineEdit.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TMultiLineEdit.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TMultiLineEdit.OnChange Event

```
property OnChange: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TMultiLineEdit.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TMultiLineEdit.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TMultiLineEdit.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TMultiLineEdit.OnEnter Event

```
property OnEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control obtains input focus.

TMultiLineEdit.OnExit Event

```
property OnExit: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control loses input focus.

TMultiLineEdit.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TMultiLineEdit.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses a key or key combination.

TMultiLineEdit.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TMultiLineEdit.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user releases a key or key combination.

TMultiLineEdit.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TMultiLineEdit.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TMultiLineEdit.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TMultiLineEdit.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TMultiLineEdit.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TMultiLineEdit.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

Available In: Visual Client Applications

This event is triggered whenever the mouse wheel is rotated forward or backward.

TMultiLineEdit.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TMultiLineEdit.OnScroll Event

```
property OnScroll: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever a scrollable control is scrolled horizontally or vertically.

TMultiLineEdit.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TMultiLineEdit.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TMultiLineEdit.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TMultiLineEdit.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TMultiLineEdit.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TMultiLineEdit.OnTouchScroll Event

```
property OnTouchScroll: TTouchScrollEvent
```

Available In: Visual Client Applications

This event is triggered whenever a touch moves in any direction over a touch-scroll-enabled control.

TMultiLineEdit.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.173 TMultiLineEditControl Component

Unit: WebEdits

Inherits From TEditControl

Available In: Visual Client Applications

The TMultiLineEditControl control is the base class for multi-line edit controls, and contains all of the core multi-line edit functionality in the form of public methods and protected properties/events that descendant classes can use to create customized multi-line edit controls.

Properties	Methods	Events

10.174 TMultipartServerRequestContent Component

Unit: WebHTTP

Inherits From TObject

Available In: Server Applications

The TMultipartServerRequestContent class is used by the TServerRequest component to represent multi-part content in a web server request.

Properties	Methods	Events
Content	Add	
Count	Clear	
	Create	
	Delete	

TMultipartServerRequestContent.Content Property

```
property Content[Index: Integer]: TServerRequestContent
```

Available In: Server Applications

Accesses the content part at the specified index.

TMultipartServerRequestContent.Count Property

```
property Count: Integer
```

Available In: Server Applications

Indicates the total number of content parts.

TMultipartServerRequestContent.Add Method

```
function Add: TServerRequestContent
```

Available In: Server Applications

Use this method to add a new content part.

TMultipartServerRequestContent.Clear Method

```
procedure Clear
```

Available In: Server Applications

Use this method to delete all content parts.

TMultipartServerRequestContent.Create Method

```
constructor Create
```

Available In: Server Applications

Use this method to create a new instance of the TMultipartServerRequestContent class.

> **Note**
> Because the multi-part content is automatically created and included as part of the TServerRequest component, you will most likely never need to call this method.

TMultipartServerRequestContent.Delete Method

```
procedure Delete(Index: Integer)
```

Available In: Server Applications

Use this method to delete the content part at the specified index.

10.175 TObjectElement Component

Unit: WebUI

Inherits From TWebElement

Available In: Visual Client Applications

The TObjectElement class is the element class for browser objects (plugins), and contains all of the browser object functionality in the form of public methods and properties/events that control classes can use to create browser object controls.

> **Note**
> This element does not provide support for objects at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events
MIMEType		
Params		

TObjectElement.MIMEType Property

```
property MIMEType: String
```

Available In: Visual Client Applications

Specifies the MIME type of the object resource that will be loaded when the URL property is specified. This information is used by the browser to determine which browser plugin to load in order to allow interaction with the resource.

An event handler can be attached to the OnLoad event to execute code when the object is loaded.

TObjectElement.Params Property

```
property Params: TStrings
```

Available In: Visual Client Applications

Specifies any parameters for the plugin that will be executed by the browser when the object resource specified by the URL property is loaded. The parameters are specified as key/value pairs:

```
Parameter=Value
```

Note
Please see the help for the applicable browser plugin in order to find out which parameters are supported by the plugin.

10.176 TObjectList Component

Unit: WebCore

Inherits From TAbstractList

Available In: Client and Server Applications

The TObjectList class is used to manage a list of class instances (objects). The constructor for the class accepts a single Boolean parameter that indicates whether the class will retain ownership of all managed objects and destroy them automatically when the TObjectList class instance is destroyed.

Properties	Methods	Events
Count	Add	
Objects	AddObjects	
OwnsObjects	Clear	
Sorted	CopyObjects	
	Create	
	Delete	
	Dequeue	
	Exchange	
	Find	
	First	
	IndexOf	
	Insert	
	Last	
	Move	
	Next	
	Pop	
	Prior	
	Push	
	Queue	
	Remove	
	Requeue	
	Sort	

TObjectList.Count Property

```
property Count: Integer
```

Available In: Client and Server Applications

Indicates the number of objects managed by the class.

TObjectList.Objects Property

```
property Objects[Index: Integer]: TObject
```

Available In: Client and Server Applications

Allows indexed access to all objects managed by the class. If the class owns its managed objects, then all managed objects are automatically destroyed when the class is destroyed.

TObjectList.OwnsObjects Property

```
property OwnsObjects: Boolean
```

Available In: Client and Server Applications

Indicates whether the class owns its managed objects and will automatically destroy all managed objects when the class is destroyed.

TObjectList.Sorted Property

```
property Sorted: Boolean
```

Available In: Client and Server Applications

Specifies that the list of objects is sorted. When the list of objects is sorted, any operations that modify the list automatically trigger a re-sort of the objects in the list.

Note
This property is only useful for TObjectList descendant classes that override two protected object comparison methods that are called in order to compare the names of objects, as well as the objects themselves.

TObjectList.Add Method

```
function Add(AObject: TObject): Integer
```

Available In: Client and Server Applications

Use this method to add an object to the list of managed objects. The object will be added to the end of the list, and the index of the object in the list will be returned.

TObjectList.AddObjects Method

```
procedure AddObjects(AList: TObjectList)
```

Available In: Client and Server Applications

Use this method to move all of the objects from a source list to the list of managed objects in this class. The source list will be empty after this method has been called.

> **Note**
> If you don't wish to delete all of the objects in the source list, use the CopyObjects method instead.

TObjectList.Clear Method

```
procedure Clear(FreeOwnedObjects: Boolean=True)
```

Available In: Client and Server Applications

Use this method to clear all object instances from the list. If the list's OwnsObjects property is True, then all instances will automatically be freed as well.

TObjectList.CopyObjects Method

```
procedure CopyObjects(AList: TObjectList)
```

Available In: Client and Server Applications

Use this method to copy all of the objects from a source list to the list of managed objects in this class.

TObjectList.Create Method

```
constructor Create

constructor Create(AOwnsObjects: Boolean)
```

Available In: Client and Server Applications

Use this method to create a new instance of the TObjectList class. This method is overloaded, and the second version of the method contains an optional AOwnsObjects parameter that indicates whether the object list instance will own its contained object instances and automatically free them when the object list itself is freed.

TObjectList.Delete Method

```
procedure Delete(Index: Integer; FreeOwnedObject: Boolean=True)
```

Available In: Client and Server Applications

Use this method to remove an object from the list of managed objects by its index. The FreeOwnedObject parameter will cause the object instance to be destroyed if the list's OwnsObjects property is True.

TObjectList.Dequeue Method

```
function Dequeue: TObject
```

Available In: Client and Server Applications

Use this method to obtain a reference to the first object (index 0) in the list of managed objects and remove the object from the list.

TObjectList.Exchange Method

```
procedure Exchange(Source: Integer; Dest: Integer)
```

Available In: Client and Server Applications

Use this method to exchange the positions of two managed objects in the list.

TObjectList.Find Method

```
function Find(const Value: String; NearestMatch: Boolean=False):
      Integer
```

Available In: Client and Server Applications

Use this method to perform a binary search of the list of objects. The Sorted property must be True or calling this method will result in an exception being raised.

Note
This method is only useful for TObjectList descendant classes that override two protected object comparison methods that are called in order to compare the names of objects, as well as the objects themselves.

TObjectList.First Method

```
function First: TObject
```

Available In: Client and Server Applications

Use this method to return the first object (index 0) in the list of managed objects.

TObjectList.IndexOf Method

```
function IndexOf(AObject: TObject): Integer
```

Available In: Client and Server Applications

Use this method to return the index of a particular object in the list of managed objects.

TObjectList.Insert Method

```
procedure Insert(Index: Integer; AObject: TObject)
```

Available In: Client and Server Applications

Use this method to insert an object at a specific index in the list of managed objects.

TObjectList.Last Method

```
function Last: TObject
```

Available In: Client and Server Applications

Use this method to return the last object (index Count-1) in the list of managed objects.

TObjectList.Move Method

```
procedure Move(Source: Integer; Dest: Integer)
```

Available In: Client and Server Applications

Use this method to move the object specified by the Source index to the position specified by the Dest index.

> **Note**
> It is important to remember that a move operation is equivalent to a delete of the object at the Source index followed by an insert of the object at the Dest index. This means that you must account for the fact that the Dest index may need to be decremented if the Source index is less than the Dest index.

TObjectList.Next Method

```
function Next(AObject: TObject; Wrap: Boolean=False): TObject
```

Available In: Client and Server Applications

Use this method to return the next object, relative to the object passed as the first parameter, in the list of managed objects. The Wrap parameter determines if the method should wrap around to the start of the list if the passed object is the last object in the list.

TObjectList.Pop Method

```
function Pop: TObject
```

Available In: Client and Server Applications

Use this method to obtain a reference to the last object (index Count-1) in the list of managed objects and remove the object from the list.

TObjectList.Prior Method

```
function Prior(AObject: TObject; Wrap: Boolean=False): TObject
```

Available In: Client and Server Applications

Use this method to return the prior object, relative to the object passed as the first parameter, in the list of managed objects. The Wrap parameter determines if the method should wrap around to the end of the list if the passed object is the first object in the list.

TObjectList.Push Method

```
procedure Push(AObject: TObject)
```

Available In: Client and Server Applications

Use this method to add an object to the end of the list of managed objects.

TObjectList.Queue Method

```
procedure Queue(AObject: TObject)
```

Available In: Client and Server Applications

Use this method to add an object to the end of the list of managed objects.

TObjectList.Remove Method

```
function Remove(AObject: TObject; FreeOwnedObject:
      Boolean=True): Integer
```

Available In: Client and Server Applications

Use this method to remove the specified object from the list of managed objects. The FreeOwnedObject parameter will cause the object instance to be destroyed if the list's OwnsObjects property is True.

TObjectList.Requeue Method

```
procedure Requeue(AObject: TObject)
```

Available In: Client and Server Applications

Use this method to insert an object at the beginning of the list of managed objects.

TObjectList.Sort Method

```
procedure Sort
```

Available In: Client and Server Applications

Use this method to sort the managed objects.

Note
This method is only useful for TObjectList descendant classes that override two protected object comparison methods that are called in order to compare the names of objects, as well as the objects themselves.

10.177 TOutsetShadow Component

Unit: WebUI

Inherits From TShadow

Available In: Visual Client Applications

The TOutsetShadow class represents the outset shadow of a UI element or control. The outset shadow appears behind the bounds of the element.

Properties	Methods	Events

10.178 TOverviewMapControlOptions Component

Unit: WebMaps

Inherits From TMapOption

Available In: Visual Client Applications

The TOverviewMapControlOptions class controls how the overview map control is configured in a TMap control. These overview map control options correspond to the overview map control options available for maps in the Google Maps API.

Properties	Methods	Events
Opened		

TOverviewMapControlOptions.Opened Property

```
property Opened: Boolean
```

Available In: Visual Client Applications

Specifies whether the overview map control is opened.

10.179 TPadding Component

Unit: WebUI

Inherits From TBoundingAttribute

Available In: Visual Client Applications

The TPadding class represents the padding within the client area of a UI element or control. The padding affects the size of the client rectangle for a UI element or control: larger padding values decrease the size of the client rectangle, while smaller padding values increase the size of the client rectangle.

Properties	Methods	Events

10.180 TPage Component

Unit: WebPages

Inherits From TControl

Available In: Visual Client Applications

The TPage component represents a page within a TPagePanel control. A page is a nested container, and can be dynamically added and removed from a page panel control. Each page instance contains a reference to a tab control via its Tab property. The properties in the tab can be modified to affect the tab caption and how the tab is sized and formatted.

Properties	Methods	Events
Background	Close	OnAnimationComplete
Border	SetActive	OnAnimationsComplete
Corners		OnCaptureEnd
Cursor		OnCaptureStart
Index		OnCapturing
InsetShadow		OnClick
Padding		OnClose
ParentPagePanel		OnCloseQuery
Tab		OnContextMenu
		OnDblClick
		OnHide
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TPage.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background for the control.

TPage.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TPage.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TPage.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TPage.Index Property

```
property Index: Integer
```

Available In: Visual Client Applications

Specifies the index of the page in its parent page panel's pages.

TPage.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Available In: Visual Client Applications

Specifies the inset shadow for the control.

TPage.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

TPage.ParentPagePanel Property

```
property ParentPagePanel: TPagePanelControl
```

Available In: Visual Client Applications

Indicates the parent page panel that contains the page.

TPage.Tab Property

```
property Tab: TTab
```

Available In: Visual Client Applications

Specifies the properties of the tab for the page.

TPage.Close Method

```
procedure Close
```

Available In: Visual Client Applications

Use this method to close the page. When this method is called, the OnCloseQuery event is triggered, followed by the OnClose event. If the OnCloseQuery event handler returns True, then the page will be hidden before the OnClose event handler is triggered. After the OnClose event handler is executed, the page will be removed from its parent page panel and disposed of.

TPage.SetActive Method

```
procedure SetActive
```

Available In: Visual Client Applications

Use this method to make the current page the active page in the parent page panel control.

TPage.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TPage.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TPage.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TPage.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TPage.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TPage.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TPage.OnClose Event

```
property OnClose: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the page is closed by the user via the tab close button, or when the Close method is called.

TPage.OnCloseQuery Event

```
property OnCloseQuery: TCloseQueryEvent
```

Available In: Visual Client Applications

This event is triggered when the page is closed by the user via the tab close button, or when the Close method is called.

Return True to allow the close to continue, or False to prevent the page from closing.

TPage.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TPage.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TPage.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TPage.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TPage.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TPage.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TPage.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TPage.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TPage.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TPage.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TPage.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TPage.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TPage.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TPage.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TPage.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.181 TPagePanel Component

Unit: WebPages

Inherits From TPagePanelControl

Available In: Visual Client Applications

The TPagePanel component represents a page panel control. A page panel control contains 0 or more instances of TPage controls that can be dynamically added and removed from the page panel control. Each page instance contains a reference to a tab control via its Tab property. The properties in the tab can be modified to affect the tab caption and how the tab is sized and formatted.

Properties	Methods	Events
Border		OnAnimationComplete
Corners		OnAnimationsComplete
Cursor		OnHide
Gutter		OnMove
Opacity		OnPageChange
Padding		OnPageChanged
PageNavigation		OnShow
TabOrder		OnSize
TabStop		
TabsVisible		

TPagePanel.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TPagePanel.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TPagePanel.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TPagePanel.Gutter Property

```
property Gutter: Integer
```

Available In: Visual Client Applications

Specifies the size, in pixels, of the blank space to show to the left of the tabs for the page panel control.

TPagePanel.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TPagePanel.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

TPagePanel.PageNavigation Property

```
property PageNavigation: Boolean
```

Available In: Visual Client Applications

Specifies whether the pages of the page panel control can be navigated using an interactive navigation bar when the TabsVisible property is False.

TPagePanel.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TPagePanel.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TPagePanel.TabsVisible Property

```
property TabsVisible: Boolean
```

Available In: Visual Client Applications

Specifies whether the tabs for the pages should be shown. The default value is True.

Note
When this property is False, the PageNavigation property can be used to show or hide an interactive navigation bar.

TPagePanel.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TPagePanel.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TPagePanel.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TPagePanel.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TPagePanel.OnPageChange Event

```
property OnPageChange: TPageChangeEvent
```

Available In: Visual Client Applications

This event is triggered whenever the ActivePage changes. To prevent the page change, return False from any event handler attached to this event.

TPagePanel.OnPageChanged Event

```
property OnPageChanged: TPageChangeEvent
```

Available In: Visual Client Applications

This event is triggered after the ActivePage has been changed.

TPagePanel.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TPagePanel.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

10.182 TPagePanelControl Component

Unit: WebPages

Inherits From TControl

Available In: Visual Client Applications

The TPagePanelControl control is the base class for page panel controls, and contains all of the core page panel functionality in the form of public methods and protected properties/events that descendant classes can use to create customized page panel controls.

Properties	Methods	Events
ActivePage	FirstPage	
PageCount	LastPage	
Pages	MakePageVisible	
	NavigateToPage	
	NewPage	
	NextPage	
	PriorPage	
	RemoveActivePage	
	ScrollNext	
	ScrollPrior	

TPagePanelControl.ActivePage Property

```
property ActivePage: TPage
```

Available In: Visual Client Applications

Specifies the active page in the page panel control.

TPagePanelControl.PageCount Property

```
property PageCount: Integer
```

Available In: Visual Client Applications

Indicates the number of pages in the page panel control.

TPagePanelControl.Pages Property

```
property Pages[AIndex: Integer]: TPage
```

Available In: Visual Client Applications

Accesses the pages in the page panel control by index.

TPagePanelControl.FirstPage Method

```
function FirstPage: TPage
```

Available In: Visual Client Applications

Use this method to make the first page in the control the active page.

TPagePanelControl.LastPage Method

```
function LastPage: TPage
```

Available In: Visual Client Applications

Use this method to make the last page in the control the active page.

TPagePanelControl.MakePageVisible Method

```
procedure MakePageVisible(APage: TPage)
```

Available In: Visual Client Applications

Use this method to ensure that the specified page's tab is visible.

TPagePanelControl.NavigateToPage Method

```
function NavigateToPage(APage: TPage): TPage
```

Available In: Visual Client Applications

Use this method to make the specified page the active page in the control.

TPagePanelControl.NewPage Method

```
function NewPage: TPage
```

Available In: Visual Client Applications

Use this method to create a new page. The new page will be positioned after all other existing pages.

TPagePanelControl.NextPage Method

```
function NextPage: TPage
```

Available In: Visual Client Applications

Use this method to make the next page, relative to the specified page, the active page in the control.

TPagePanelControl.PriorPage Method

```
function PriorPage: TPage
```

Available In: Visual Client Applications

Use this method to make the prior page, relative to the specified page, the active page in the control.

TPagePanelControl.RemoveActivePage Method

```
procedure RemoveActivePage
```

Available In: Visual Client Applications

Use this method to remove the active page, if one exists. If the ActivePage property is nil, then this method does nothing.

TPagePanelControl.ScrollNext Method

```
procedure ScrollNext
```

Available In: Visual Client Applications

Use this method to scroll the page tabs to the right by one tab.

TPagePanelControl.ScrollPrior Method

```
procedure ScrollPrior
```

Available In: Visual Client Applications

Use this method to scroll the page tabs to the left by one tab.

10.183 TPaint Component

Unit: WebPaint

Inherits From TControl

Available In: Visual Client Applications

The TPaint component represents a drawing/painting control. A paint control can be used for general drawing, charting, animation, and more. It contains a reference to a TCanvasElement instance that can be used to perform all drawing operations, and automatically resizes the canvas instance to match the dimensions of the control.

Properties	Methods	Events
Canvas		OnAnimationComplete
Cursor		OnCaptureEnd
Hint		OnCaptureStart
Opacity		OnCapturing
		OnClick
		OnContextMenu
		OnDblClick
		OnHide
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TPaint.Canvas Property

```
property Canvas: TCanvasElement
```

Available In: Visual Client Applications

This property provdes access to a TCanvasElement instance that can be used to perform all drawing operations within the dimensions of the control.

TPaint.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TPaint.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TPaint.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TPaint.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TPaint.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TPaint.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TPaint.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TPaint.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TPaint.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TPaint.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TPaint.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TPaint.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TPaint.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TPaint.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TPaint.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TPaint.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TPaint.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TPaint.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TPaint.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TPaint.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TPaint.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TPaint.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TPaint.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.184 TPanControlOptions Component

Unit: WebMaps

Inherits From TMapOption

Available In: Visual Client Applications

The TPanControlOptions class controls how the pan control is configured in a TMap control. These pan control options correspond to the pan control options available for maps in the Google Maps API.

Properties	Methods	Events
Position		

TPanControlOptions.Position Property

```
property Position: TMapControlPosition
```

Available In: Visual Client Applications

Specifies the position of the pan control.

10.185 TPanel Component

Unit: WebCtnrs

Inherits From TPanelControl

Available In: Visual Client Applications

The TPanel component represents a panel control with a border and a caption bar with minimize/restore and close buttons.

Properties	Methods	Events
ActivateOnClick		OnAnimationComplete
AutoSize		OnAnimationsComplete
Background		OnCaptionBarDblClick
Border		OnCaptureEnd
CaptionBar		OnCaptureStart
Client		OnCapturing
Corners		OnClick
Cursor		OnClose
Hint		OnCloseQuery
Opacity		OnContextMenu
OutsetShadow		OnDblClick
ScrollBars		OnHide
ScrollSupport		OnKeyDown
TabOrder		OnKeyPress
TabStop		OnKeyUp
		OnMinimize
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMouseWheel
		OnMove
		OnRestore
		OnScroll
		OnShow
		OnSize
		OnTouchCancel

		OnTouchEnd
		OnTouchMove
		OnTouchScroll
		OnTouchStart

TPanel.ActivateOnClick Property

```
property ActivateOnClick: Boolean
```

Available In: Visual Client Applications

Specifies whether the control should automatically be brought to the front when it, or any child controls, are clicked.

TPanel.AutoSize Property

```
property AutoSize: TAutoSize
```

Available In: Visual Client Applications

Specifies how (if at all) the control should automatically be sized based upon the child controls placed in the panel.

TPanel.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

TPanel.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TPanel.CaptionBar Property

```
property CaptionBar: TPanelCaptionBar
```

Available In: Visual Client Applications

Specifies the properties of the caption bar for the control.

TPanel.Client Property

```
property Client: TPanelClient
```

Available In: Visual Client Applications

Specifies the properties of the client area for the control.

TPanel.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TPanel.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TPanel.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TPanel.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TPanel.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Available In: Visual Client Applications

Specifies the outset shadow for the control.

TPanel.ScrollBars Property

```
property ScrollBars: TScrollBars
```

Available In: Visual Client Applications

Specifies which scrollbars to show, if any.

Note
Even if this property is set to sbHorizontal, sbVertical, or sbBoth, a scrollbar will only be shown if the size of the contents and/or the child controls of the control exceed the client rectangle for the control.

TPanel.ScrollSupport Property

```
property ScrollSupport: TScrollSupport
```

Available In: Visual Client Applications

Specifies the directions in which the control can be scrolled, if any.

> **Note**
> This property only applies to scrolling via touch and mouse wheel movements, and does not apply to programmatic scrolling or scrolling via the scroll bars themselves.

TPanel.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TPanel.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TPanel.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TPanel.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TPanel.OnCaptionBarDblClick Event

```
property OnCaptionBarDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the caption bar is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TPanel.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TPanel.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TPanel.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TPanel.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TPanel.OnClose Event

```
property OnClose: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the panel is closed by the user via the caption bar close button, or when the Close method is called.

TPanel.OnCloseQuery Event

```
property OnCloseQuery: TCloseQueryEvent
```

Available In: Visual Client Applications

This event is triggered when the panel is closed by the user via the caption bar close button, or when the Close method is called. Return True to allow the close to continue, or False to prevent the panel from closing.

TPanel.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TPanel.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TPanel.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TPanel.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and the user presses a key or key combination.

TPanel.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and presses/releases a key or key combination.

TPanel.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and the user releases a key or key combination.

TPanel.OnMinimize Event

```
property OnMinimize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the panel is minimized.

TPanel.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TPanel.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TPanel.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TPanel.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TPanel.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TPanel.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

Available In: Visual Client Applications

This event is triggered whenever the mouse wheel is rotated forward or backward.

TPanel.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TPanel.OnRestore Event

```
property OnRestore: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the panel is restored from a minimized state.

TPanel.OnScroll Event

```
property OnScroll: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever a scrollable control is scrolled horizontally or vertically.

TPanel.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TPanel.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TPanel.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TPanel.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TPanel.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TPanel.OnTouchScroll Event

```
property OnTouchScroll: TTouchScrollEvent
```

Available In: Visual Client Applications

This event is triggered whenever a touch moves in any direction over a touch-scroll-enabled control.

TPanel.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.186 TPanelCaptionBar Component

Unit: WebCtnrs

Inherits From TCaptionBarControl

Available In: Visual Client Applications

The TPanelCaptionBar component represents the caption bar for a TPanel control, and includes a caption and minimize/restore and close buttons.

Properties	Methods	Events
Alignment		
AllowClose		
AllowMinimize		
AllowMove		
Background		
Caption		
Cursor		
Font		
Icon		
Padding		

TPanelCaptionBar.Alignment Property

```
property Alignment: TContentAlignment
```

Available In: Visual Client Applications

Specifies the alignment of the caption in the caption bar.

TPanelCaptionBar.AllowClose Property

```
property AllowClose: Boolean
```

Available In: Visual Client Applications

Specifies whether the close button should be shown in the caption bar.

TPanelCaptionBar.AllowMinimize Property

```
property AllowMinimize: Boolean
```

Available In: Visual Client Applications

Specifies whether the minimize/restore button should be shown in the caption bar.

TPanelCaptionBar.AllowMove Property

```
property AllowMove: Boolean
```

Available In: Visual Client Applications

Specifies whether the user can press and hold a mouse or touch on the caption bar and drag the container panel to a new position.

TPanelCaptionBar.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

TPanelCaptionBar.Caption Property

```
property Caption: String
```

Available In: Visual Client Applications

Specifies the caption to display in the caption bar.

TPanelCaptionBar.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TPanelCaptionBar.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TPanelCaptionBar.Icon Property

```
property Icon: TIconProperties
```

Available In: Visual Client Applications

Specifies the properties of the icon used with the caption bar.

TPanelCaptionBar.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

10.187 TPanelClient Component

Unit: WebCtnrs

Inherits From TComponent

Available In: Visual Client Applications

The TPanelClient component represents the client area for a TPanel control.

Properties	Methods	Events
Background		
InsetShadow		
Padding		

TPanelClient.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background for the control.

TPanelClient.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Available In: Visual Client Applications

Specifies the inset shadow for the control.

TPanelClient.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

10.188 TPanelControl Component

Unit: WebCtnrs

Inherits From TScrollableControl

Available In: Visual Client Applications

The TPanelControl control is the base class for panel controls, and contains all of the core panel functionality in the form of public methods and protected properties/events that descendant classes can use to create customized panel controls.

Properties	Methods	Events
	Close	

TPanelControl.Close Method

```
procedure Close
```

Available In: Visual Client Applications

Use this method to close the panel. When this method is called, the OnCloseQuery event is triggered, followed by the OnClose event. If the OnCloseQuery event handler returns True, then the panel will be hidden before the OnClose event is triggered.

10.189 TParser Component

Unit: WebCore

Inherits From TObject

Available In: Client and Server Applications

The TParser class is a parser class for parsing JSON strings, and is used for loading forms, control interfaces, and datasets (columns, rows, and transaction operations). It can be used as a general-purpose JSON parser in your applications.

Properties	Methods	Events
PropertyNameType	CheckString	
Token	CheckSymbol	
TokenLiteral	CheckToken	
TokenPos	ErrorIfNotSkipString	
TokenString	ErrorIfNotSkipSymbol	
	ErrorIfNotSkipToken	
	ErrorIfNotString	
	ErrorIfNotSymbol	
	ErrorIfNotToken	
	ExpectedError	
	GetBoolean	
	GetFloat	
	GetInteger	
	GetString	
	Initialize	
	NextToken	
	SkipArray	
	SkipObject	
	SkipProperty	
	SkipPropertyValue	
	SkipString	
	SkipSymbol	
	SkipToken	

TParser.PropertyNameType Property

```
property PropertyNameType: Char
```

Available In: Client and Server Applications

Specifies the Token type to use when parsing JSON object property names. The default value is tkString, which means that the parser will expect property names to be specified in the same way as strings: enclosed in double-quote (") characters.

TParser.Token Property

```
property Token: Char
```

Available In: Client and Server Applications

Indicates the current token character or token type. The following special token types are used for non-character tokens:

Token Type	Description
tkTerm	Indicates that the current token is the terminating character (#0)
tkSymbol	Indicates that the current token is a symbol
tkString	Indicates that the current token is a string enclosed in double-quote (") characters
tkInteger	Indicates that the current token is an integer
tkFloat	Indicates that the current token is a floating-point number

TParser.TokenLiteral Property

```
property TokenLiteral: String
```

Available In: Client and Server Applications

Indicates the current token in its literal form. For strings, this means that the double quote (") characters are included in the value returned by this property.

TParser.TokenPos Property

```
property TokenPos: Integer
```

Available In: Client and Server Applications

Indicates the current token position (1-based) in the string being parsed.

TParser.TokenString Property

```
property TokenString: String
```

Available In: Client and Server Applications

Indicates the current token. For strings, this means that the double quote (") characters are **not** included in the value returned by this property.

TParser.CheckString Method

```
function CheckString(const Value: String): Boolean
```

Available In: Client and Server Applications

Use this method to determine if the current token is a string.

TParser.CheckSymbol Method

```
function CheckSymbol(const Value: String): Boolean
```

Available In: Client and Server Applications

Use this method to determine if the current token is a symbol.

TParser.CheckToken Method

```
function CheckToken(Value: Char): Boolean
```

Available In: Client and Server Applications

Use this method to determine if the current token is a particular character or type of token.

TParser.ErrorIfNotSkipString Method

```
procedure ErrorIfNotSkipString(const Value: String)
```

Available In: Client and Server Applications

Use this method to determine if the current token is a string and, if so, to move to the next token. If the current token is not a string, then an exception will be raised.

TParser.ErrorIfNotSkipSymbol Method

```
procedure ErrorIfNotSkipSymbol(const Value: String)
```

Available In: Client and Server Applications

Use this method to determine if the current token is a symbol and, if so, to move to the next token. If the current token is not a symbol, then an exception will be raised.

TParser.ErrorIfNotSkipToken Method

```
procedure ErrorIfNotSkipToken(Value: Char)
```

Available In: Client and Server Applications

Use this method to determine if the current token is a particular character or type of token and, if so, to move to the next token. If the current token is not the particular character or type of token, then an exception will be raised.

TParser.ErrorIfNotString Method

```
procedure ErrorIfNotString(const Value: String)
```

Available In: Client and Server Applications

Use this method to determine if the current token is a string. If the current token is not a string, then an exception will be raised.

TParser.ErrorIfNotSymbol Method

```
procedure ErrorIfNotSymbol(const Value: String)
```

Available In: Client and Server Applications

Use this method to determine if the current token is a symbol. If the current token is not a symbol, then an exception will be raised.

TParser.ErrorIfNotToken Method

```
procedure ErrorIfNotToken(Value: Char)
```

Available In: Client and Server Applications

Use this method to determine if the current token is a particular character or type of token. If the current token is not the particular character or type of token, then an exception will be raised.

TParser.ExpectedError Method

```
procedure ExpectedError(const Value: String)
```

Available In: Client and Server Applications

Use this method to raise an exception due to a parsing error, such as a case when a certain token literal was expected and a different literal was found instead.

TParser.GetBoolean Method

```
function GetBoolean: Boolean
```

Available In: Client and Server Applications

Use this method to retrieve the current token as a boolean value.

> **Note**
> If the Token property is not equal to tkSymbol and the TokenString property is not a valid JSON boolean literal (true, false), an exception will be raised.

TParser.GetFloat Method

```
function GetFloat: Double
```

Available In: Client and Server Applications

Use this method to retrieve the current token as a float value.

Note
If the Token property is not equal to tkFloat or tkInteger, an exception will be raised.

TParser.GetInteger Method

```
function GetInteger: Integer
```

Available In: Client and Server Applications

Use this method to retrieve the current token as an integer value.

> **Note**
> If the Token property is not equal to tkInteger, an exception will be raised.

TParser.GetString Method

```
function GetString: String
```

Available In: Client and Server Applications

Use this method to retrieve the current token as a string value, without any enclosing double-quote (") characters.

> **Note**
> If the Token property is not equal to tkString, an exception will be raised.

TParser.Initialize Method

```
procedure Initialize(const TextToParse: String; FullNumbers:
      Boolean=True; APropertyNameType: Char=tkString)
```

Available In: Client and Server Applications

Use this method to initialize the parser with a string containing the text that is to be parsed. This method will automatically "seed" the parser by calling the NextToken method.

TParser.NextToken Method

```
procedure NextToken
```

Available In: Client and Server Applications

Use this method to move to the next token.

TParser.SkipArray Method

```
function SkipArray: Boolean
```

Available In: Client and Server Applications

Use this method to skip over a JSON array. If the current token is not the left bracket ([) character, then this method will return False.

TParser.SkipObject Method

```
function SkipObject: Boolean
```

Available In: Client and Server Applications

Use this method to skip over a JSON object. If the current token is not the left brace ({) character, then this method will return False.

TParser.SkipProperty Method

```
procedure SkipProperty
```

Available In: Client and Server Applications

Use this method to skip over an entire JSON property, including the property name and value. The SkipPropertyValue method is used to skip over the value.

TParser.SkipPropertyValue Method

```
procedure SkipPropertyValue
```

Available In: Client and Server Applications

Use this method to skip over a JSON property value. If the JSON property value is an object, then the SkipObject method is used to skip over the value. If the JSON property value is an array, then the SkipArray method is used to skip over the value. Otherwise, the NextToken method is used to skip over the value.

TParser.SkipString Method

```
function SkipString(const Value: String): Boolean
```

Available In: Client and Server Applications

Use this method to determine if the current token is a string and, if so, to move to the next token.

TParser.SkipSymbol Method

```
function SkipSymbol(const Value: String): Boolean
```

Available In: Client and Server Applications

Use this method to determine if the current token is a symbol and, if so, to move to the next token.

TParser.SkipToken Method

```
function SkipToken(Value: Char): Boolean
```

Available In: Client and Server Applications

Use this method to determine if the current token is a particular character or type of token and, if so, to move to the next token.

10.190 TPasswordEdit Component

Unit: WebEdits

Inherits From TEditControl

Available In: Visual Client Applications

The TPasswordEdit component represents a password edit control. An edit control allows the user to directly enter an input value using the keyboard but, instead of showing the input value in the edit control, the input characters are masked so that they cannot be seen.

Properties	Methods	Events
Alignment		OnAnimationComplete
Cursor		OnAnimationsComplete
DataColumn		OnCaptureEnd
DataSet		OnCaptureStart
Direction		OnCapturing
Enabled		OnChange
Font		OnClick
Hint		OnContextMenu
MaxLength		OnDblClick
ReadOnly		OnEnter
TabOrder		OnExit
TabStop		OnHide
Text		OnKeyDown
		OnKeyPress
		OnKeyUp
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TPasswordEdit.Alignment Property

```
property Alignment: TContentAlignment
```

Available In: Visual Client Applications

Specifies the alignment of the input value for the control.

TPasswordEdit.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TPasswordEdit.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TPasswordEdit.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

TPasswordEdit.Direction Property

```
property Direction: TContentDirection
```

Available In: Visual Client Applications

Specifies the direction in which the text is displayed/edited.

TPasswordEdit.Enabled Property

```
property Enabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TPasswordEdit.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TPasswordEdit.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TPasswordEdit.MaxLength Property

```
property MaxLength: Integer
```

Available In: Visual Client Applications

Specifies the maximum allowable length, in characters, of the Text property for the control. A value of 0 specifies an unlimited allowable length.

TPasswordEdit.ReadOnly Property

```
property ReadOnly: Boolean
```

Available In: Visual Client Applications

Specifies whether the control's input value can be modified by the user. The default value is False.

Note
The input value can always be programmatically modified.

TPasswordEdit.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TPasswordEdit.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TPasswordEdit.Text Property

```
property Text: String
```

Available In: Visual Client Applications

Specifies the control's input value as a string.

TPasswordEdit.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TPasswordEdit.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TPasswordEdit.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TPasswordEdit.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TPasswordEdit.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TPasswordEdit.OnChange Event

```
property OnChange: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TPasswordEdit.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TPasswordEdit.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TPasswordEdit.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TPasswordEdit.OnEnter Event

```
property OnEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control obtains input focus.

TPasswordEdit.OnExit Event

```
property OnExit: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control loses input focus.

TPasswordEdit.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TPasswordEdit.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses a key or key combination.

TPasswordEdit.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TPasswordEdit.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user releases a key or key combination.

TPasswordEdit.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TPasswordEdit.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TPasswordEdit.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TPasswordEdit.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TPasswordEdit.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TPasswordEdit.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TPasswordEdit.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TPasswordEdit.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TPasswordEdit.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TPasswordEdit.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TPasswordEdit.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TPasswordEdit.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.191 TPasswordInputElement Component

Unit: WebUI

Inherits From TInputElement

Available In: Visual Client Applications

The TPasswordInputElement class is the base element class for password input elements, and contains all of the password input functionality in the form of public methods and properties/events that control classes can use to create password input controls.

> **Note**
> This element does not provide support for password input elements at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events

10.192 TPersistent Component

Unit: WebCore

Inherits From TObject

Available In: Client and Server Applications

The TPersistent class is the base class for classes that require the ability to save/load their published properties to/from JSON strings. All of the functionality in the TPersistent class is protected and normally only accessible to descendant classes. The TWriter class is used by TPersistent descendants for outputting properties to a JSON string, and the TReader class for parsing properties from a JSON string.

All TComponent, TCollection, TCollectionItem, and TControl classes are TPersistent-descendant classes, and all contain functionality for automatically loading their published properties. In addition, the TFormControl class contains special functionality for loading the published properties for a form instance and all of its contained components/controls. This functionality is used to load the published properties for a form class when an instance of the class is created.

Properties	Methods	Events
	Assign	
	Load	
	Save	

TPersistent.Assign Method

```
procedure Assign(APersistent: TPersistent)
```

Available In: Client and Server Applications

Use this method to assign the specified source TPersistent instance to the current TPersistent instance.

> **Note**
> If a descendant class does not override this method and the specified source TPersistent instance is not nil, the default behavior is for this method to call the protected TPersistent AssignTo method of the source TPersistent instance, passing the current instance as the parameter. The default behavior of the AssignTo method is to do nothing.

TPersistent.Load Method

```
procedure Load(AReader: TReader)
```

Available In: Client and Server Applications

Use this method to load the current TPersistent instance using the specified TReader instance.

TPersistent.Save Method

```
procedure Save(AWriter: TWriter)
```

Available In: Client and Server Applications

Use this method to save the current TPersistent instance using the specified TWriter instance.

10.193 TPersistentStorage Component

Unit: WebComps

Inherits From TObject

Available In: Client Applications

The TPersistentStorage object encapsulates the HTML5 local storage functionality, which allows the application to store name-value pairs of strings using the host web browser's storage facilities. This type of storage is preferable to cookies (TCookies) due to the fact that cookies are normally limited to around 4k of storage.

> **Note**
> The component library includes two global instances of this class called LocalStorage and SessionStorage in the WebComps unit that should be used instead of creating new instances of the class. The LocalStorage instance represents the persistent local storage in the host web browser that is available across browser instances/sessions, whereas the SessionStorage instance represents the session-only storage in the host web browser that is cleared whenever the browser is closed.

Properties	Methods	Events
Count	Clear	OnChange
Items	ClearAll	
Keys	Exists	
	Set	

TPersistentStorage.Count Property

```
property Count: Integer
```

Available In: Client Applications

Indicates the number of items defined.

TPersistentStorage.Items Property

```
property Items[Index: Integer]: String

property Items[const AName: String]: String
```

Available In: Client Applications

Allows access to all defined items, either by index or by name.

TPersistentStorage.Keys Property

```
property Keys[Index: Integer]: String
```

Available In: Client Applications

Allows access to all defined item names by index.

TPersistentStorage.Clear Method

```
procedure Clear(const AName: String)
```

Available In: Client Applications

Use this method to clear the specified item.

> **Note**
> Use the Exists method to determine if a item exists before trying to clear the item.

TPersistentStorage.ClearAll Method

```
procedure ClearAll
```

Available In: Client Applications

Use this method to clear all items.

TPersistentStorage.Exists Method

```
function Exists(const AName: String): Boolean
```

Available In: Client Applications

Use this method to determine if the specified item exists.

TPersistentStorage.Set Method

```
procedure Set(const AName: String; const Value: String)
```

Available In: Client Applications

Use this method to set an item value. The Name parameter is the item name and the Value parameter is the item value.

TPersistentStorage.OnChange Event

```
property OnChange: TStorageChangeEvent
```

Available In: Client Applications

This event is triggered whenever the contents of the persistent local storage is changed by a different session in the browser.

> **Note**
> This event is only triggered for the persistent local storage and not the session-only local storage.

10.194 TPlugin Component

Unit: WebBrwsr

Inherits From TWebControl

Available In: Visual Client Applications

The TPlugin component represents a web browser plugin container control that can interact with any type of object resource that is supported by a registered plugin in the web browser, including MP3/MP4 audio files, PDF document files, and MPEG video files.

Properties	Methods	Events
Background		OnAnimationComplete
Border		OnAnimationsComplete
Corners		OnHide
Cursor		OnLoad
DataColumn		OnMove
DataSet		OnShow
Hint		OnSize
InsetShadow		OnUnload
Loaded		
MIMEType		
Opacity		
OutsetShadow		
Padding		
Params		
URL		

TPlugin.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

TPlugin.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TPlugin.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TPlugin.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TPlugin.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TPlugin.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

TPlugin.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TPlugin.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Available In: Visual Client Applications

Specifies the inset shadow for the control.

TPlugin.Loaded Property

```
property Loaded: Boolean
```

Available In: Visual Client Applications

Indicates whether the object resource specified by the URL property has been loaded.

An event handler can be attached to the OnLoad event to execute code when the object resource is loaded.

TPlugin.MIMEType Property

```
property MIMEType: String
```

Available In: Visual Client Applications

Specifies the MIME type of the object resource that will be loaded when the URL property is specified. This information is used by the browser to determine which browser plugin to load in order to allow interaction with the resource.

An event handler can be attached to the OnLoad event to execute code when the object is loaded.

TPlugin.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TPlugin.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Available In: Visual Client Applications

Specifies the outset shadow for the control.

TPlugin.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

TPlugin.Params Property

```
property Params: TStrings
```

Available In: Visual Client Applications

Specifies any parameters for the plugin that will be executed by the browser when the object resource specified by the URL property is loaded. The parameters are specified as key/value pairs:

```
Parameter=Value
```

Note
Please see the help for the applicable browser plugin in order to find out which parameters are supported by the plugin.

TPlugin.URL Property

```
property URL: String
```

Available In: Visual Client Applications

Specifies the URL for the object resource. Whenever the URL property changes, the OnUnload event is triggered immediately. The OnLoad event is triggered once the object resource has been loaded.

TPlugin.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TPlugin.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TPlugin.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TPlugin.OnLoad Event

```
property OnLoad: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the file specified by the URL property has been completely loaded by a browser plugin.

TPlugin.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TPlugin.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TPlugin.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TPlugin.OnUnload Event

```
property OnUnload: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the currently-loaded file specified by the URL property has been unloaded.

10.195 TPoint Component

Unit: WebUI

Inherits From TObject

Available In: Client Applications

The TPoint class represents the X/Y integer coordinates of a point. It is used with the TElement class and descendant classes for calculating coordinates during event management for an element, or elements.

Properties	Methods	Events
X	Assign	
Y	Clear	
	Create	
	Equals	
	Offset	

TPoint.X Property

```
property X: Integer
```

Available In: Client Applications

Specifies the horizontal position of the point.

TPoint.Y Property

```
property Y: Integer
```

Available In: Client Applications

Specifies the vertical position of the point.

TPoint.Assign Method

```
procedure Assign(APoint: TPoint)

procedure Assign(AX,AY: Integer)
```

Available In: Client Applications

Use this method to assign a source point to the point instance.

TPoint.Clear Method

```
procedure Clear
```

Available In: Client Applications

Use this method to set both of the point coordinates to 0.

TPoint.Create Method

```
constructor Create(AX,AY: Integer)
```

Available In: Client Applications

Use this method to create a new instance of the TPoint class using the provided X and Y coordinates.

TPoint.Equals Method

```
function Equals(APoint: TPoint): Boolean
```

Available In: Client Applications

Use this method to test if two points have the same coordinates.

TPoint.Offset Method

```
procedure Offset(AX,AY: Integer)
```

Available In: Client Applications

Use this method to offset the point. Offsetting a point increments or decrements its X and Y properties using the AX and AY parameters, respectively.

10.196 TProgressBar Component

Unit: WebProgs

Inherits From TControl

Available In: Visual Client Applications

The TProgressBar component represents a progress bar control for showing visual progress between a minimum and maximum set of values.

Properties	Methods	Events
Background		OnAnimationComplete
Border		OnAnimationsComplete
Corners		OnCaptureEnd
Cursor		OnCaptureStart
Indicator		OnCapturing
MaxValue		OnClick
MinValue		OnContextMenu
Padding		OnDblClick
Position		OnHide
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TProgressBar.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background for the control.

TProgressBar.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TProgressBar.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TProgressBar.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TProgressBar.Indicator Property

```
property Indicator: TProgressBarIndicator
```

Available In: Visual Client Applications

Accesses the properties of the progress bar indicator.

TProgressBar.MaxValue Property

```
property MaxValue: Integer
```

Available In: Visual Client Applications

Specifies the maximum progress bar value. The default value is 100.

TProgressBar.MinValue Property

```
property MinValue: Integer
```

Available In: Visual Client Applications

Specifies the minimum progress bar value. The default value is 0.

TProgressBar.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

TProgressBar.Position Property

```
property Position: Integer
```

Available In: Visual Client Applications

Specifies the current progress bar position between the MinValue and MaxValue properties.

TProgressBar.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TProgressBar.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TProgressBar.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TProgressBar.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TProgressBar.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TProgressBar.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TProgressBar.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TProgressBar.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TProgressBar.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TProgressBar.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TProgressBar.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TProgressBar.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TProgressBar.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TProgressBar.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TProgressBar.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TProgressBar.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TProgressBar.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TProgressBar.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TProgressBar.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TProgressBar.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TProgressBar.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.197 TProgressBarIndicator Component

Unit: WebProgs

Inherits From TComponent

Available In: Visual Client Applications

The TProgressBarIndicator component represents the indicator portion of a TProgressBar control.

Properties	Methods	Events
Background		
Border		
Corners		

TProgressBarIndicator.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

TProgressBarIndicator.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TProgressBarIndicator.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

10.198 TProgressDialog Component

Unit: WebForms

Inherits From TDialogControl

Available In: Visual Client Applications

The TProgressDialog component represents a progress dialog control. Please see the Showing Progress Dialogs topic for more information on using progress dialogs.

Properties	Methods	Events
Corners		OnAnimationComplete
Cursor		OnAnimationsComplete
Message		OnCaptureEnd
Opacity		OnCaptureStart
OutsetShadow		OnCapturing
		OnClick
		OnClose
		OnContextMenu
		OnDblClick
		OnHide
		OnKeyDown
		OnKeyPress
		OnKeyUp
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMouseWheel
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TProgressDialog.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TProgressDialog.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TProgressDialog.Message Property

```
property Message: String
```

Available In: Visual Client Applications

Specifies the message to display in the dialog.

TProgressDialog.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TProgressDialog.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Available In: Visual Client Applications

Specifies the outset shadow for the control.

TProgressDialog.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TProgressDialog.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TProgressDialog.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TProgressDialog.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TProgressDialog.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TProgressDialog.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TProgressDialog.OnClose Event

```
property OnClose: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the dialog's Close method is called.

TProgressDialog.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TProgressDialog.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TProgressDialog.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TProgressDialog.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and the user presses a key or key combination.

TProgressDialog.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and presses/releases a key or key combination.

TProgressDialog.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and the user releases a key or key combination.

TProgressDialog.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TProgressDialog.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TProgressDialog.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TProgressDialog.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TProgressDialog.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TProgressDialog.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

Available In: Visual Client Applications

This event is triggered whenever the mouse wheel is rotated forward or backward.

TProgressDialog.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TProgressDialog.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TProgressDialog.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TProgressDialog.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TProgressDialog.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TProgressDialog.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TProgressDialog.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.199 TRadioButton Component

Unit: WebBtns

Inherits From TStateButtonControl

Available In: Visual Client Applications

The TRadioButton component represents a radio button control. A radio button control is a state control that is used in groups to allow the user to select a specific control by using a mouse click, or by pushing the spacebar or enter key. A set of radio button controls are considered in the same group when they share the same parent container. You can change the position of the radio button controls within a container by modifying their LayoutOrder property.

Properties	Methods	Events
AutoWidth		OnAnimationComplete
Caption		OnAnimationsComplete
Cursor		OnCaptureEnd
DataColumn		OnCaptureStart
DataSet		OnCapturing
Enabled		OnChange
Font		OnClick
Hint		OnContextMenu
ReadOnly		OnEnter
SelectionState		OnExit
TabOrder		OnHide
TabStop		OnKeyDown
ValueSelected		OnKeyPress
		OnKeyUp
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TRadioButton.AutoWidth Property

```
property AutoWidth: Boolean
```

Available In: Visual Client Applications

Specifies whether the width of the radio button should be automatically set based upon the Caption and Font properties.

TRadioButton.Caption Property

```
property Caption: String
```

Available In: Visual Client Applications

Specifies the caption for the control.

Note
The caption area of a control is also clickable with the mouse or touch interface.

TRadioButton.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TRadioButton.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TRadioButton.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

TRadioButton.Enabled Property

```
property Enabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TRadioButton.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TRadioButton.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TRadioButton.ReadOnly Property

```
property ReadOnly: Boolean
```

Available In: Visual Client Applications

Specifies whether the control's input value can be modified by the user. The default value is False.

> **Note**
> The input value can always be programmatically modified.

TRadioButton.SelectionState Property

```
property SelectionState: TSelectionState
```

Available In: Visual Client Applications

Specifies the selection state of the control.

Note
The ssIndeterminate selection state can only be set programmatically. When the user toggles the selection for the control, it will alternate between the ssSelected and ssUnselected selection states.

TRadioButton.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TRadioButton.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TRadioButton.ValueSelected Property

```
property ValueSelected: String
```

Available In: Visual Client Applications

Specifies the textual value to use for the selected state when reading and writing data to and from the data column that the control is bound to. The default value is the string representation of the layout order of the control within its parent container control.

TRadioButton.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TRadioButton.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TRadioButton.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TRadioButton.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TRadioButton.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TRadioButton.OnChange Event

```
property OnChange: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TRadioButton.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TRadioButton.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TRadioButton.OnEnter Event

```
property OnEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control obtains input focus.

TRadioButton.OnExit Event

```
property OnExit: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control loses input focus.

TRadioButton.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TRadioButton.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses a key or key combination.

TRadioButton.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TRadioButton.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when the control has input focus and the user releases a key or key combination.

TRadioButton.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TRadioButton.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TRadioButton.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TRadioButton.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TRadioButton.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TRadioButton.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TRadioButton.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TRadioButton.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TRadioButton.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TRadioButton.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TRadioButton.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TRadioButton.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.200 TRasterImage Component

Unit: WebSrvr

Inherits From TExternalObject

Available In: Server Applications

The TRasterImage class is used to load and manipulate raster images in server applications. It allows you to load/save raster images in bitmap, JPEG, GIF, PNG, and TIFF formats, as well as resize such images in order to create thumbnail images.

Properties	Methods	Events
ColorSpace	Create	
Format	Resize	
HasAlphaChannel	SaveToFile	
HasRealDPI	SaveToStream	
HasRealPixelSize		
HasTranslucent		
Height		
HorzResolution		
IsPartiallyScalable		
IsScalable		
MIMEType		
PixelFormat		
VertResolution		
Width		

TRasterImage.ColorSpace Property

```
property ColorSpace: TColorSpace
```

Available In: Server Applications

Indicates the color space of the loaded image.

TRasterImage.Format Property

```
property Format: TImageFormat
```

Available In: Server Applications

Indicates the format of the loaded image.

TRasterImage.HasAlphaChannel Property

```
property HasAlphaChannel: Boolean
```

Available In: Server Applications

Indicates whether the loaded image has an alpha channel.

TRasterImage.HasRealDPI Property

```
property HasRealDPI: Boolean
```

Available In: Server Applications

Indicates whether the loaded image has DPI information.

TRasterImage.HasRealPixelSize Property

```
property HasRealPixelSize: Boolean
```

Available In: Server Applications

Indicates whether the loaded image has pixel size information.

TRasterImage.HasTranslucent Property

```
property HasTranslucent: Boolean
```

Available In: Server Applications

Indicates whether the loaded image has alpha values that are not just 0 (transparent) or 255 (opaque).

TRasterImage.Height Property

```
property Height: Integer
```

Available In: Server Applications

Indicates the height of the loaded image.

TRasterImage.HorzResolution Property

```
property HorzResolution: Double
```

Available In: Server Applications

Indicates the horizontal resolution (DPI) of the loaded image.

TRasterImage.IsPartiallyScalable Property

```
property IsPartiallyScalable: Boolean
```

Available In: Server Applications

Indicates whether the loaded image is partially scalable.

TRasterImage.IsScalable Property

```
property IsScalable: Boolean
```

Available In: Server Applications

Indicates whether the loaded image is scalable.

TRasterImage.MIMEType Property

```
property MIMEType: String
```

Available In: Server Applications

Indicates the MIME type of loaded image.

TRasterImage.PixelFormat Property

```
property PixelFormat: TPixelFormat
```

Available In: Server Applications

Indicates the pixel format of the loaded image.

TRasterImage.VertResolution Property

```
property VertResolution: Double
```

Available In: Server Applications

Indicates the vertical resolution (DPI) of the loaded image.

TRasterImage.Width Property

```
property Width: Integer
```

Available In: Server Applications

Indicates the width of the loaded image.

TRasterImage.Create Method

```
constructor Create(const FileName: String;
      UseEmbeddedColorManagement: Boolean=False)

constructor Create(Stream: TStream; UseEmbeddedColorManagement:
      Boolean=False)
```

Available In: Server Applications

Use this method to create a new instance of the TRasterImage class. You can create a new instance and load an existing image using a file name or a TStream descendant class instance.

Note
The specified file name should contain an absolute path. Relative paths are permitted, but they rely on the current working directory. The current working directory is a per-process setting and is not reliable in a multi-threaded setting like the web server.

TRasterImage.Resize Method

```
function Resize(NewWidth: Integer; NewHeight: Integer):
      TRasterImage
```

Available In: Server Applications

Use this method to create a new TRasterImage instance that is the resized version of the loaded image.

TRasterImage.SaveToFile Method

```
procedure SaveToFile(const FileName: String; ImageFormat:
      TImageFormat=ifPNG)
```

Available In: Server Applications

Use this method to save the loaded image to the specified file.

Note
The specified file name should contain an absolute path. Relative paths are permitted, but they rely on the current working directory. The current working directory is a per-process setting and is not reliable in a multi-threaded setting like the web server.

TRasterImage.SaveToStream Method

```
procedure SaveToStream(Stream: TStream; ImageFormat:
      TImageFormat=ifPNG)
```

Available In: Server Applications

Use this method to save the loaded image to the specified stream.

10.201 TReader Component

Unit: WebCore

Inherits From TObject

Available In: Client and Server Applications

The TReader class is a class used by TPersistent-descendant classes to load their published properties from JSON strings, and is used for loading forms and control interfaces. It can be used as a general-purpose JSON reader in your applications.

When a TReader instance is created, the constructor allows you to specify the date-time format to use when reading date-time properties.

Properties	Methods	Events
GlobalComponent	BeginArray	
Level	BeginObject	
RootComponent	Create	
	EndArray	
	EndObject	
	EndOfArray	
	EndOfObject	
	ErrorIfNotMoreArrayElements	
	ErrorIfNotMoreProperties	
	ErrorIfNotPropertyName	
	GetPropertyName	
	GetPropertyValue	
	Initialize	
	IsArray	
	IsBoolean	
	IsNull	
	IsObject	
	IsString	
	MoreArrayElements	
	MoreProperties	
	ReadBoolean	
	ReadDateTime	
	ReadFloat	
	ReadInteger	
	ReadPropertyName	

	ReadString	
	ReadStrings	
	SkipArrayElement	
	SkipProperty	
	SkipPropertyName	
	SkipPropertySeparator	
	SkipPropertyValue	

TReader.GlobalComponent Property

```
property GlobalComponent: TComponent
```

Available In: Client and Server Applications

Specifies an optional global component to use with the reader. This property is used with forms and databases to determine how to apply any component reference fixups that need to occur after the form or database is loaded. The default value is nil.

Note
If this property is not specified, then the Owner property of the RootComponent will be used as the instance to use for applying component reference fixups.

TReader.Level Property

```
property Level: Integer
```

Available In: Client and Server Applications

Indicates the current nesting level for any JSON objects and/or arrays. Whenever the TReader class begins to read an object or array using the BeginObject or BeginArray methods, the nesting level is incremented. Whenever the EndObject or EndArray methods are called, the nesting level is decremented.

TReader.RootComponent Property

```
property RootComponent: TComponent
```

Available In: Client and Server Applications

Specifies an optional root component to use with the reader. The root component is the component that is supposed to be the owner of all TPersistent instances being read from the incoming JSON string. This property is used with form controls to specify the owner of all component instances created during the loading of the form. The default value is nil.

TReader.BeginArray Method

```
procedure BeginArray
```

Available In: Client and Server Applications

Use this method to begin reading an array. If the current token in the incoming JSON string is not a left bracket ([), an exception will be raised.

TReader.BeginObject Method

```
procedure BeginObject
```

Available In: Client and Server Applications

Use this method to begin reading an object. If the current token in the incoming JSON string is not a left brace ({), an exception will be raised.

TReader.Create Method

```
constructor Create(ADateTimeFormat: TDateTimeFormat=dtfRaw)
```

Available In: Client and Server Applications

Use this method to create a new instance of the TReader class. The optional ADateTimeFormat parameter indicates whether date and time values should be handled as an ISO 8601 date and time string value, or as a raw Unix date and time integer value (the number of milliseconds since midnight, January 1, 1970).

TReader.EndArray Method

```
procedure EndArray
```

Available In: Client and Server Applications

Use this method to end reading an array. If the current token in the incoming JSON string is not a right bracket (]), an exception will be raised.

TReader.EndObject Method

```
procedure EndObject
```

Available In: Client and Server Applications

Use this method to end reading an object. If the current token in the incoming JSON string is not a right brace (}), an exception will be raised.

TReader.EndOfArray Method

```
function EndOfArray: Boolean
```

Available In: Client and Server Applications

Use this method to determine if the current token in the incoming JSON string is a right bracket (]).

TReader.EndOfObject Method

```
function EndOfObject: Boolean
```

Available In: Client and Server Applications

Use this method to determine if the current token in the incoming JSON string is a right brace (}).

TReader.ErrorIfNotMoreArrayElements Method

```
procedure ErrorIfNotMoreArrayElements
```

Available In: Client and Server Applications

Use this method to check for more array elements using the MoreArrayElements method and raise an exception if the result is False.

TReader.ErrorIfNotMoreProperties Method

```
procedure ErrorIfNotMoreProperties
```

Available In: Client and Server Applications

Use this method to check for more properties using the MoreProperties method and raise an exception if the result is False.

TReader.ErrorIfNotPropertyName Method

```
procedure ErrorIfNotPropertyName(const Value: String)
```

Available In: Client and Server Applications

Use this method to read a property name using the GetPropertyName method and raise an exception if the property name does not match the specified property name parameter.

TReader.GetPropertyName Method

```
function GetPropertyName: String
```

Available In: Client and Server Applications

Use this method to read a property name, without any enclosing double-quote (") characters (if applicable).

TReader.GetPropertyValue Method

```
function GetPropertyValue: String
```

Available In: Client and Server Applications

Use this method to read a property value, without any enclosing double-quote (") characters (if applicable).

TReader.Initialize Method

```
procedure Initialize(const Value: String; ACompressedProperties:
      Boolean=False)
```

Available In: Client and Server Applications

Use this method to initialize the reader with a JSON string to read.

TReader.IsArray Method

```
function IsArray: Boolean
```

Available In: Client and Server Applications

Use this method to determine if the current token in the incoming JSON string is a left bracket ([).

TReader.IsBoolean Method

```
function IsBoolean: Boolean
```

Available In: Client and Server Applications

Use this method to determine if the current token in the incoming JSON string is a boolean value (true or false).

TReader.IsNull Method

```
function IsNull: Boolean
```

Available In: Client and Server Applications

Use this method to determine if the current token in the incoming JSON string is a null literal.

TReader.IsObject Method

```
function IsObject: Boolean
```

Available In: Client and Server Applications

Use this method to determine if the current token in the incoming JSON string is a left brace ({).

TReader.IsString Method

```
function IsString: Boolean
```

Available In: Client and Server Applications

Use this method to determine if the current token in the incoming JSON string is a string value ("").

TReader.MoreArrayElements Method

```
function MoreArrayElements: Boolean
```

Available In: Client and Server Applications

Use this method to determine if the current token in the incoming JSON string is a comma (,).

TReader.MoreProperties Method

```
function MoreProperties: Boolean
```

Available In: Client and Server Applications

Use this method to determine if the current token in the incoming JSON string is a comma (,).

TReader.ReadBoolean Method

```
function ReadBoolean: Boolean
```

Available In: Client and Server Applications

Use this method to read a boolean value. If the current token in the incoming JSON string is not a valid JSON boolean literal (true, false), an exception will be raised.

TReader.ReadDateTime Method

```
function ReadDateTime: DateTime
```

Available In: Client and Server Applications

Use this method to read a date-time value. How a date-time value is read is controlled by the TDateTimeFormat parameter in the TReader class constructor.

TReader.ReadFloat Method

```
function ReadFloat: Double
```

Available In: Client and Server Applications

Use this method to read a float value. If the current token in the incoming JSON string is not a valid JSON float or integer literal, an exception will be raised.

TReader.ReadInteger Method

```
function ReadInteger: Integer
```

Available In: Client and Server Applications

Use this method to read an integer value. If the current token in the incoming JSON string is not a valid JSON integer literal, an exception will be raised.

TReader.ReadPropertyName Method

```
function ReadPropertyName: String

procedure ReadPropertyName(const Value: String)
```

Available In: Client and Server Applications

Use this method to read a property name using the GetPropertyName and then call the SkipPropertyName and SkipPropertySeparator methods to skip over the property name and property separator.

> **Note**
> The version of this method that accepts a property name as a parameter calls the ErrorIfNotPropertyName method instead of the GetPropertyName method.

TReader.ReadString Method

```
function ReadString: String
```

Available In: Client and Server Applications

Use this method to read a string value, without any enclosing double-quote (") characters. If the current token in the incoming JSON string is not a valid JSON string literal, an exception will be raised.

TReader.ReadStrings Method

```
procedure ReadStrings(Value: TStrings; KeyValue: Boolean=False)
```

Available In: Client and Server Applications

Use this method to read the strings for a TStrings class instance. The KeyValue parameter determines how the strings are read:

- If the KeyValue parameter is True, then the strings are read as JSON property values from a series of JSON properties and populated as <Key>=<Value> strings in the TStrings instance.
- If the KeyValue parameter is False, then the strings are read as a JSON array property and populated as normal strings in the TStrings instance.

TReader.SkipArrayElement Method

```
procedure SkipArrayElement
```

Available In: Client and Server Applications

Use this method to skip over a JSON array element. The TPersistent class uses this method to skip an array element when the class instance does not know how to properly load the array element.

TReader.SkipProperty Method

```
procedure SkipProperty
```

Available In: Client and Server Applications

Use this method to skip over a JSON object property. The TPersistent class uses this method to skip a property when the class instance does not know how to properly load the property.

TReader.SkipPropertyName Method

```
procedure SkipPropertyName
```

Available In: Client and Server Applications

Use this method to skip over a JSON object property name. The TPersistent class uses this method to skip past the property name after reading the name and determining that the property exists in the class for the instance being loaded.

TReader.SkipPropertySeparator Method

```
procedure SkipPropertySeparator
```

Available In: Client and Server Applications

Use this method to skip over a JSON object property separator (:). The TPersistent class uses this method to skip past the property name/separator after reading the property name and determining that the property exists in the class for the instance being loaded.

TReader.SkipPropertyValue Method

```
procedure SkipPropertyValue
```

Available In: Client and Server Applications

Use this method to skip over a JSON object property value. The TPersistent class uses this method to skip past any property values that it cannot load for any reason.

10.202 TRect Component

Unit: WebUI

Inherits From TObject

Available In: Client Applications

The TRect class represents a rectangle using integer coordinates. It is used with the TElement class and descendant classes for calculating rectangle coordinates for all of the element's dimensional functionality, such as layout management.

Properties	Methods	Events
Bottom	Anchor	
Empty	Assign	
Height	Clear	
Left	Contains	
Right	Create	
Top	Equals	
Width	Fit	
	Inflate	
	Interpolate	
	Normalize	
	Offset	
	Scale	
	Union	

TRect.Bottom Property

```
property Bottom: Integer
```

Available In: Client Applications

Specifies the bottom Y coordinate of the rectangle.

TRect.Empty Property

```
property Empty: Boolean
```

Available In: Client Applications

Indicates whether the rectangle is empty. A rectangle is considered empty if its width or height is 0.

TRect.Height Property

```
property Height: Integer
```

Available In: Client Applications

Indicates the height of the rectangle (read-only).

TRect.Left Property

```
property Left: Integer
```

Available In: Client Applications

Specifies the left X coordinate of the rectangle.

TRect.Right Property

```
property Right: Integer
```

Available In: Client Applications

Specifies the right X coordinate of the rectangle.

TRect.Top Property

```
property Top: Integer
```

Available In: Client Applications

Specifies the top Y coordinate of the rectangle.

TRect.Width Property

```
property Width: Integer
```

Available In: Client Applications

Indicates the width of the rectangle (read-only).

TRect.Anchor Method

```
procedure Anchor
```

Available In: Client Applications

Use this method to anchor the rectangle. Anchoring a rectangle changes its Left and Top properties to 0, and adjusts the Right and Bottom properties accordingly so that the rectangle retains its same Width and Height.

TRect.Assign Method

```
procedure Assign(ARect: TRect)

procedure Assign(ALeft,ATop,ARight,ABottom: Integer)
```

Available In: Client Applications

Use this method to assign a source rectangle, or source rectangle coordinates, to the rectangle instance.

TRect.Clear Method

```
procedure Clear
```

Available In: Client Applications

Use this method to set all of the rectangle coordinates to 0.

TRect.Contains Method

```
function Contains(X,Y: Integer): Boolean

function Contains(APoint: TPoint): Boolean
```

Available In: Client Applications

Use this method to determine if the specified coordinates are contained within the coordinates of the rectangle.

TRect.Create Method

```
constructor Create(ALeft,ATop,ARight,ABottom: Integer=0)
```

Available In: Client Applications

Use this method to create a new instance of the TRect class. The optional ALeft, ATop, ARight, and ABottom parameters will initialize the instance with the provided coordinates.

TRect.Equals Method

```
function Equals(ARect: TRect): Boolean
```

Available In: Client Applications

Use this method to test if two rectangles have the same coordinates.

TRect.Fit Method

```
procedure Fit(AWidth,AHeight: Integer)
```

Available In: Client Applications

Use this method to proportionally adjust the width and height of the rectangle so that it fits within the specified AWidth and AHeight parameters.

TRect.Inflate Method

```
procedure Inflate(ALeft,ATop,ARight,ABottom: Integer)
```

Available In: Client Applications

Use this method to inflate the rectangle. Inflating a rectangle increments or decrements its Left, Top, Right, and Bottom properties using the specified parameters.

TRect.Interpolate Method

```
procedure Interpolate(ARect: TRect; AAmount: Double)
```

Available In: Client Applications

Use this method to calculate a new rectangle using the difference between the coordinates of the rectangle and a source rectangle multiplied by a specified distance amount. This calculation is useful for moving a rectangle along a given linear path between a source rectangle and a destination rectangle.

TRect.Normalize Method

```
procedure Normalize
```

Available In: Client Applications

Use this method to normalize the rectangle. Normalizing a rectangle changes (if necessary) its Left, Top, Right, and Bottom properties so that the Left property is less than or equal to the Right property and the Top property is less than or equal to the Bottom property.

TRect.Offset Method

```
procedure Offset(ALeft: Integer; ATop: Integer)
```

Available In: Client Applications

Use this method to offset the rectangle. Offsetting a rectangle increments or decrements its Left and Top properties using the ALeft and ATop parameters, respectively.

TRect.Scale Method

```
procedure Scale(ARatio: Double)
```

Available In: Client Applications

Use this method to proportionally scale the coordinates of the rectangle using the specified ARatio parameter.

TRect.Union Method

```
procedure Union(ARect: TRect)
```

Available In: Client Applications

Use this method to union the coordinates of the rectangle with another rectangle. A union operation is a max operation on each pair of coordinates, taking the smallest of the two rectangles' Left and Top properties, and the largest of the two rectangles' Right and Bottom properties.

10.203 TRepeatControl Component

Unit: WebCtrls

Inherits From TControl

Available In: Visual Client Applications

The TRepeatControl control is the base class for controls that have a repeatable click behavior, and contains all of the repeatable click functionality in the form of public methods and protected properties/events that descendant classes can use to create customized controls.

Properties	Methods	Events

10.204 TRequestHandler Component

Unit: WebRequest

Inherits From TComponent

Available In: Server Applications

The TRequestHandler component is the class used for web server request handlers, and contains all of the core request handler functionality in the form of properties/events and methods that can be used by a server application to handle an incoming web server request.

Properties	Methods	Events
	HandleRequest	OnCreate
		OnDestroy
		OnHandleRequest

TRequestHandler.HandleRequest Method

```
procedure HandleRequest(Request: TWebServerRequest)
```

Available In: Server Applications

Use this method to manually call the OnHandleRequest event handler.

> **Note**
> Normally this method is only called by the web server when routing an incoming web server request into a server application.

TRequestHandler.OnCreate Event

```
property OnCreate: TNotifyEvent
```

Available In: Server Applications

This event is triggered after the request handler is created and initialized.

Note
Any design-time components placed on the request handler will already be instantiated and initialized before this event is triggered.

TRequestHandler.OnDestroy Event

```
property OnDestroy: TNotifyEvent
```

Available In: Server Applications

This event is triggered before the request handler is destroyed. Use this event to dispose of any instances or resources that may have been allocated in the OnCreate event handler.

TRequestHandler.OnHandleRequest Event

```
property OnHandleRequest: THandleRequestEvent
```

Available In: Server Applications

This event is triggered when an incoming web server request has been routed to the server application and needs to be handled. The Request parameter will contain the web server request that needs to be handled.

10.205 TRotateControlOptions Component

Unit: WebMaps

Inherits From TMapOption

Available In: Visual Client Applications

The TRotateControlOptions class controls how the rotate control is configured in a TMap control. These rotate control options correspond to the rotate control options available for maps in the Google Maps API.

Properties	Methods	Events
Position		

TRotateControlOptions.Position Property

```
property Position: TMapControlPosition
```

Available In: Visual Client Applications

Specifies the position of the rotate control.

10.206 TScript Component

Unit: WebComps

Inherits From TComponent

Available In: Client Applications

The TScript component represents a dynamically-loaded script and is used to introduce external JavaScript into the global execution context from a URL, as opposed to being linked via the emitted HTML with the application.

Properties	Methods	Events
URL		OnError
		OnLoad

TScript.URL Property

```
property URL: String
```

Available In: Client Applications

Specifies the full URL for the script to be loaded dynamically. Whenever this property is changed, the existing script is removed from the global execution context and the new script is downloaded from the specified URL. Once the script has been downloaded and loaded, the OnLoad event will be triggered. If there was an error downloading the script, then the OnError event will be triggered.

TScript.OnError Event

```
property OnError: TNotifyEvent
```

Available In: Client Applications

This event is triggered when the download of the script encounters an error.

TScript.OnLoad Event

```
property OnLoad: TNotifyEvent
```

Available In: Client Applications

This event is triggered when the download of the script is complete and the script is loaded into the application's global execution context.

10.207 TScrollableControl Component

Unit: WebCtrls

Inherits From TControl

Available In: Visual Client Applications

The TScrollableControl control is the base class for scrollable controls, and contains all of the scrolling functionality in the form of public methods and protected properties/events that descendant classes can use to create customized scrollable controls.

Properties	Methods	Events

10.208 TScrollPanel Component

Unit: WebCtnrs

Inherits From TScrollPanelControl

Available In: Visual Client Applications

The TScrollPanel component represents a scrollable panel control. A scrollable panel control is useful when you need only a portion of a form or other type of container control to be scrollable.

Properties	Methods	Events
ActivateOnClick		OnAnimationComplete
Background		OnAnimationsComplete
Client		OnCaptureEnd
Cursor		OnCaptureStart
Hint		OnCapturing
Opacity		OnClick
OutsetShadow		OnClientSize
ScrollBars		OnContextMenu
ScrollSupport		OnDblClick
TabOrder		OnHide
TabStop		OnKeyDown
		OnKeyPress
		OnKeyUp
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMouseWheel
		OnMove
		OnScroll
		OnScrollSize
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchScroll

		OnTouchStart

TScrollPanel.ActivateOnClick Property

```
property ActivateOnClick: Boolean
```

Available In: Visual Client Applications

Specifies whether the control should automatically be brought to the front when it, or any child controls, are clicked.

TScrollPanel.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

TScrollPanel.Client Property

```
property Client: TScrollPanelClient
```

Available In: Visual Client Applications

Specifies the properties of the client area for the control.

TScrollPanel.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TScrollPanel.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TScrollPanel.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TScrollPanel.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Available In: Visual Client Applications

Specifies the outset shadow for the control.

TScrollPanel.ScrollBars Property

```
property ScrollBars: TScrollBars
```

Available In: Visual Client Applications

Specifies which scrollbars to show, if any.

Note
Even if this property is set to sbHorizontal, sbVertical, or sbBoth, a scrollbar will only be shown if the size of the contents and/or the child controls of the control exceed the client rectangle for the control.

TScrollPanel.ScrollSupport Property

```
property ScrollSupport: TScrollSupport
```

Available In: Visual Client Applications

Specifies the directions in which the control can be scrolled, if any.

> **Note**
> This property only applies to scrolling via touch and mouse wheel movements, and does not apply to programmatic scrolling or scrolling via the scroll bars themselves.

TScrollPanel.TabOrder Property

```
property TabOrder: Integer
```

Available In: Visual Client Applications

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TScrollPanel.TabStop Property

```
property TabStop: Boolean
```

Available In: Visual Client Applications

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TScrollPanel.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TScrollPanel.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TScrollPanel.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TScrollPanel.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TScrollPanel.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TScrollPanel.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TScrollPanel.OnClientSize Event

```
property OnClientSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the width and/or height of the client area of the control is changed.

TScrollPanel.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TScrollPanel.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TScrollPanel.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TScrollPanel.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and the user presses a key or key combination.

TScrollPanel.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and presses/releases a key or key combination.

TScrollPanel.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

Available In: Visual Client Applications

This event is triggered when a child control has input focus and the user releases a key or key combination.

TScrollPanel.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TScrollPanel.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TScrollPanel.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TScrollPanel.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TScrollPanel.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TScrollPanel.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

Available In: Visual Client Applications

This event is triggered whenever the mouse wheel is rotated forward or backward.

TScrollPanel.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TScrollPanel.OnScroll Event

```
property OnScroll: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever a scrollable control is scrolled horizontally or vertically.

TScrollPanel.OnScrollSize Event

```
property OnScrollSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the width and/or height of the scrollable area of the control is changed.

TScrollPanel.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TScrollPanel.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TScrollPanel.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TScrollPanel.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TScrollPanel.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TScrollPanel.OnTouchScroll Event

```
property OnTouchScroll: TTouchScrollEvent
```

Available In: Visual Client Applications

This event is triggered whenever a touch moves in any direction over a touch-scroll-enabled control.

TScrollPanel.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.209 TScrollPanelClient Component

Unit: WebCtnrs

Inherits From TComponent

Available In: Visual Client Applications

The TScrollPanelClient component represents the client area for a TScrollPanel control.

Properties	Methods	Events
Background		
InsetShadow		
Padding		

TScrollPanelClient.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background for the control.

TScrollPanelClient.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Available In: Visual Client Applications

Specifies the inset shadow for the control.

TScrollPanelClient.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

10.210 TScrollPanelControl Component

Unit: WebCtnrs

Inherits From TScrollableControl

Available In: Visual Client Applications

The TScrollPanelControl control is the base class for scroll panel controls, and contains all of the core scroll panel functionality in the form of public methods and protected properties/events that descendant classes can use to create customized scroll panel controls.

Properties	Methods	Events

10.211 TServerRequest Component

Unit: WebHTTP

Inherits From TComponent

Available In: Client and Server Applications

The TServerRequest component represents a dynamic HTTP request to the web server from which the application was loaded or, provided that the web server allows it, a different origin (protocol, host, and port).

> **Note**
> Web server requests are executed asynchronously in client browser applications and synchronously in server applications.

Properties	Methods	Events
CompleteURL	Cancel	OnCancel
ConnectTimeout	Execute	OnComplete
Content	ParseXML	OnError
ContentStream	Reset	OnProgress
CrossOriginCredentials		OnStart
Headers		OnTimeout
Method		
MethodLiteral		
MultipartContent		
Params		
Password		
Queue		
ResponseContent		
ResponseContentStream		
ResponseContentType		
ResponseHeaders		
ResponseType		
StatusCode		
StatusText		
Timeout		
URL		
UserAgent		
UserName		

TServerRequest.CompleteURL Property

```
property CompleteURL: String
```

Available In: Client and Server Applications

Indicates the complete URL that will be used with the server request, including any query parameters specified via the Params property.

TServerRequest.ConnectTimeout Property

```
property ConnectTimeout: Integer
```

Available In: Server Applications

Specifies the connection timeout for the web server request. A value of 0 specifies that the server request should wait indefinitely. The default value is 30 seconds.

Note
Client applications use the Timeout property for both the connection timeout and the request timeout, whereas server applications only use this property for the connection timeout and the Timeout property for the timeout for the request itself.

TServerRequest.Content Property

```
property Content: TStrings
```

Available In: Client and Server Applications

Specifies the content to send with the web server request.

TServerRequest.ContentStream Property

```
property ContentStream: TStream
```

Available In: Server Applications

Specifies a reference to a TStream descendant class instance to send as the content with the web server request.

Warning
This is just a reference to the TStream descendant class instance, so please make sure that you do not free this class instance before the TServerRequest Execute method completes.

TServerRequest.CrossOriginCredentials Property

```
property CrossOriginCredentials: Boolean
```

Available In: Client and Server Applications

Specifies whether HTTP cookies and/or HTTP authentication headers are sent with any requests to an origin (protocol, host, and port) that is different than the origin for the application. The default value is False.

TServerRequest.Headers Property

```
property Headers: TStrings
```

Available In: Client and Server Applications

Specifies any additional headers to be sent with the web server request.

> **Note**
> All necessary headers will be automatically populated by the web browser, so only use this property for specifying headers that are necessary for the web server request, such as a **Content-Type** header when sending content to the web server in the Content property.

TServerRequest.Method Property

```
property Method: TServerRequestMethod
```

Available In: Client and Server Applications

Specifies the HTTP method to use for the web server request.

TServerRequest.MethodLiteral Property

```
property MethodLiteral: String
```

Available In: Client and Server Applications

Indicates the actual HTTP method literal specified by the Method property.

TServerRequest.MultipartContent Property

```
property MultipartContent: TMultipartServerRequestContent
```

Available In: Server Applications

Specifies the multi-part content to send with the web server request.

TServerRequest.Params Property

```
property Params: TStrings
```

Available In: Client and Server Applications

Specifies the URL parameters for the web server request. The Params property is automatically set up to use the equals (=) character for separating the name/value pairs, so you can use the Values property to set them by name. If specifying the parameters directly as name=value strings in the string list, each parameter should be specified as a separate string.

TServerRequest.Password Property

```
property Password: String
```

Available In: Client and Server Applications

Specifies the password to use for authenticating the request with the web server using basic HTTP authentication, if required by the web server for the URL specified for the request. Both the UserName and the Password property are required in order to properly authenticate the request.

> **Note**
> Basic HTTP authentication is a legacy, header-based mechanism used for authentication and is not the same as the authentication API used by the web server.

TServerRequest.Queue Property

```
property Queue: TServerRequestQueue
```

Available In: Client and Server Applications

If the current server request instance was created by a TServerRequestQueue component, then this property will contain a reference to the instance of the queue component that created the request.

TServerRequest.ResponseContent Property

```
property ResponseContent: TStrings
```

Available In: Client and Server Applications

Indicates any content that was returned from the web server in the response to the server request.

> **Note**
> This property will only be populated once the OnComplete event has been triggered.

TServerRequest.ResponseContentStream Property

```
property ResponseContentStream: TStream
```

Available In: Server Applications

Contains any content included with the response to the server request as a TStream descendant class instance when the ResponseContentType is set to srStream. The use of a TStream interface makes it easier to work with response content that is binary and not textual.

TServerRequest.ResponseContentType Property

```
property ResponseContentType: String
```

Available In: Client and Server Applications

Indicates the value of the **Content-Type** HTTP header, if present, that was returned from the web server in the response to the server request. If no **Content-Type** header was returned, then this property will return an empty string ('').

TServerRequest.ResponseHeaders Property

```
property ResponseHeaders: TStrings
```

Available In: Client and Server Applications

Indicates any headers that were returned from the web server in the response to the server request.

> **Note**
> This property will only be populated once the OnComplete event has been triggered.

TServerRequest.ResponseType Property

```
property ResponseType: TServerRequestResponseType
```

Available In: Server Applications

Specifies how any content included with the response to the server request should be represented. The default value is srText.

TServerRequest.StatusCode Property

```
property StatusCode: Integer
```

Available In: Client and Server Applications

Indicates the status code returned by the web server in response to the server request.

> **Note**
> This property will only be populated once the OnComplete event has been triggered.

TServerRequest.StatusText Property

```
property StatusText: String
```

Available In: Client and Server Applications

Indicates the status message returned by the web server in response to the server request.

> **Note**
> This property will only be populated once the OnComplete event has been triggered.

TServerRequest.Timeout Property

```
property Timeout: Integer
```

Available In: Client and Server Applications

Specifies how long the server request should wait, in seconds, for a successful connection to the server before returning an error. A value of 0 specifies that the server request should wait indefinitely. The default value is 30 seconds.

Note
Client applications use this property for both the connection timeout and the request timeout, whereas server applications only use this property for the request timeout and the ConnectTimeout property for the connection timeout.

TServerRequest.URL Property

```
property URL: TServerRequestURL
```

Available In: Client and Server Applications

Specifies the URL of the resource that the server request wishes to get data from, or send data to.

Warning
It is highly recommended that you only use relative URLs only in this property. Most modern web browsers will prevent server requests that don't access resources from the same origin (protocol, host, port number) from executing, unless the web server specifically allows such a request. Such a request is referred to as a cross-origin resource sharing request. For more information on how to permit such requests with the web server, please see the Configuring the Web Server topic.

TServerRequest.UserAgent Property

```
property UserAgent: String
```

Available In: Server Applications

Specifies the user agent to use with the server request. Some public APIs require that you set the **User-Agent** header with any server requests to the API, and setting this property will ensure that the **User-Agent** header is set to the specified value. The default value is "".

TServerRequest.UserName Property

```
property UserName: String
```

Available In: Client and Server Applications

Specifies the user name to use for authenticating the request with the web server using basic HTTP authentication, if required by the web server for the URL specified for the request. Both the UserName and the Password property are required in order to properly authenticate the request.

Note
Basic HTTP authentication is a legacy, header-based mechanism used for authentication and is not the same as the authentication API used by the web server.

TServerRequest.Cancel Method

```
procedure Cancel(KeepExecuting: Boolean=True)
```

Available In: Client and Server Applications

Use this method to abort a pending web server request.

TServerRequest.Execute Method

```
procedure Execute
```

Available In: Client and Server Applications

Use this method to execute a web server request after specifying the Method and URL properties, at a minimum.

TServerRequest.ParseXML Method

```
function ParseXML: TDocument
```

Available In: Client and Server Applications

Use this method to parse XML content returned as a response to the web server request. This method returns a TDocument web browser DOM object instance that can be used to manipulate the various nodes of the XML document as DOM elements.

Please see the WebDOM unit for the various interfaces to the DOM objects such as TDocument available in the web browser.

TServerRequest.Reset Method

```
procedure Reset
```

Available In: Client and Server Applications

Use this method to reset all server request properties back to their default values. This is useful for situations where a server request instance is being reused multiple times and you need to be sure that the server request is initialized back to its default state.

TServerRequest.OnCancel Event

```
property OnCancel: TServerRequestEvent
```

Available In: Client and Server Applications

This event is triggered when the server request has been cancelled using the Cancel method.

> **Note**
> This event is triggered before the OnComplete event.

TServerRequest.OnComplete Event

```
property OnComplete: TServerRequestEvent
```

Available In: Client and Server Applications

This event is triggered when the server request is complete. Use the StatusCode property to determine the status code returned by the web server and, subsequently, whether the request was successful or not.

TServerRequest.OnError Event

```
property OnError: TServerRequestErrorEvent
```

Available In: Client and Server Applications

This event is triggered when the server request has encountered an error.

> **Note**
> This event is triggered before the OnComplete event.

TServerRequest.OnProgress Event

```
property OnProgress: TServerRequestProgressEvent
```

Available In: Client and Server Applications

This event is triggered as the server request is executed and represents the current progress of the request.

> **Note**
> The Current and Total parameters passed to the event handler are determined by the browser completing the server request and may differ between browsers.

TServerRequest.OnStart Event

```
property OnStart: TServerRequestEvent
```

Available In: Client and Server Applications

This event is triggered when the server request is started. A server request is started when the Execute method is called.

TServerRequest.OnTimeout Event

```
property OnTimeout: TServerRequestEvent
```

Available In: Client and Server Applications

This event is triggered when the server request has timed out.

> **Note**
> This event is triggered before the OnComplete event.

10.212 TServerRequestContent Component

Unit: WebHTTP

Inherits From TObject

Available In: Server Applications

The TServerRequestContent class is used by the TMultipartServerRequestContent class to represent one part of the multi-part content for a web server request.

Properties	Methods	Events
Content	Create	
ContentStream		
Headers		

TServerRequestContent.Content Property

```
property Content: String
```

Available In: Server Applications

Specifies the textual content to include as the content part. This property and the ContentStream property are mutually-exclusive. If both properties are assigned a value, then the ContentStream property will take precedence.

> **Note**
> If this property is assigned a value and the **Content-Type** header is not specified using the Headers property, it will be automatically set to "application/json; charset=utf-8" when the server request is executed.

TServerRequestContent.ContentStream Property

```
property ContentStream: TStream
```

Available In: Server Applications

Specifies a TStream descendant class instance that contains the binary content to include as the content part. This property and the Content property are mutually-exclusive. If both properties are assigned a value, then the ContentStream property will take precedence.

Note
If this property is assigned a value and the **Content-Type** header is not specified using the Headers property, it will be automatically set to "application/octet-stream" when the server request is executed.

Warning
This is just a reference to the TStream descendant class instance, so please make sure that you do not free this class instance before the TServerRequest Execute method is called.

TServerRequestContent.Headers Property

```
property Headers: TStrings
```

Available In: Server Applications

Provides access to the headers for the content part.

TServerRequestContent.Create Method

```
constructor Create
```

Available In: Server Applications

Use this method to create a new instance of the TServerRequestContent class.

10.213 TServerRequestQueue Component

Unit: WebHTTP

Inherits From TComponent

Available In: Client Applications

The TServerRequestQueue component represents a queue of dynamic HTTP requests to the web server from which the application was loaded or, provided that the web server allows it, a different origin (protocol, host, and port). Serialization allows server requests to be executed in a specific order. If executed individually, such requests would normally be executed asynchronously by the web browser and would not have a predictable completion order.

Properties	Methods	Events
NumPendingRequests	AddRequest	
	CancelAllRequests	
	CancelRequest	
	ExecuteRequests	
	GetNewRequest	
	NextPendingRequest	

TServerRequestQueue.NumPendingRequests Property

```
property NumPendingRequests: Integer
```

Available In: Client Applications

Indicates the number of requests currently awaiting execution.

> **Note**
> The value returned by this property does **not** include the currently-executing request, if one exists. If a currently-executing request fails for any reason, then the value returned by this property **will** include the failed request when this property is referenced from an OnComplete event handler.

TServerRequestQueue.AddRequest Method

```
procedure AddRequest(Value: TServerRequest)
```

Available In: Client Applications

Use this method to add a new request to the end of the queue of web server requests. When using this method, always use the GetNewRequest to obtain a new request that can be modified before calling this method.

TServerRequestQueue.CancelAllRequests Method

```
procedure CancelAllRequests
```

Available In: Client Applications

Use this method to cancel all pending web server requests in the queue.

TServerRequestQueue.CancelRequest Method

```
procedure CancelRequest(KeepExecuting: Boolean=True)
```

Available In: Client Applications

Use this method to cancel the currently executing web server request in the queue. After the current request is cancelled, the next request in the queue will be executed.

TServerRequestQueue.ExecuteRequests Method

```
procedure ExecuteRequests
```

Available In: Client Applications

Use this method to execute any pending web server requests in the queue.

TServerRequestQueue.GetNewRequest Method

```
function GetNewRequest: TServerRequest
```

Available In: Client Applications

Use this method to allocate a new web server request to use with the queue. After modifying the request as required, call the AddRequest method to add the request to the queue.

TServerRequestQueue.NextPendingRequest Method

```
function NextPendingRequest: TServerRequest
```

Available In: Client Applications

Use this method to get access to the next pending web server request in the queue.

10.214 TServerSession Component

Unit: WebSession

Inherits From TComponent

Available In: Client Applications

The TServerSession component represents a web server session and is used for authentication and retrieving basic information about the authenticated user after authentication has taken place. In addition, the TDatabase component uses a TServerSession component instance for performing authentication as necessary when making database API requests. The server session is associated with the TDatabase component using its ServerSession property.

> **Note**
> Eventually this component will provide the classes and methods for remotely administering a web server instance, but currently does not provide such functionality. At this time, the only way to remotely administer a web server is by using the IDE.

Properties	Methods	Events
AdministrationResource	Authenticate	OnAuthenticate
ApplicationsResource	CancelRequest	OnComplete
Authenticated	Deauthenticate	OnProgress
Authenticating	HasPrivilege	
AuthenticationResource	HasRole	
BaseURL	RefreshEffectiveAccess	
DatabasesResource		
DebuggerResource		
EffectivePrivileges		
EffectiveRoles		
FullName		
KeepAliveResource		
ModulesResource		
Password		
RequestExecuting		
RequestStatusCode		
RequestStatusText		
UserName		

TServerSession.AdministrationResource Property

```
property AdministrationResource: String
```

Available In: Client Applications

Specifies the resource name that should be used with any administration API requests.

TServerSession.ApplicationsResource Property

```
property ApplicationsResource: String
```

Available In: Client Applications

Specifies the resource name that should be used with any server application API requests.

TServerSession.Authenticated Property

```
property Authenticated: Boolean
```

Available In: Client Applications

Indicates whether the current session is authenticated (from the perspective of the client application).

TServerSession.Authenticating Property

```
property Authenticating: Boolean
```

Available In: Client Applications

Indicates whether the current session is in the process of authenticating.

TServerSession.AuthenticationResource Property

```
property AuthenticationResource: String
```

Available In: Client Applications

Specifies the resource name that should be used with any authentication API requests.

TServerSession.BaseURL Property

```
property BaseURL: String
```

Available In: Client Applications

Specifies the base URL for the target web server for any server requests made by the server session. In addition, this base URL is used by the TDatabase component to create the proper URLs for making database API requests. The server session is associated with the TDatabase component using its ServerSession property.

TServerSession.DatabasesResource Property

```
property DatabasesResource: String
```

Available In: Client Applications

Specifies the resource name that should be used with any database API requests.

TServerSession.DebuggerResource Property

```
property DebuggerResource: String
```

Available In: Client Applications

Specifies the resource name that should be used with any debugger API requests.

TServerSession.EffectivePrivileges Property

```
property EffectivePrivileges: TStringList
```

Available In: Client Applications

Contains a list of the effective privileges for the user when the Authenticated property is True.

TServerSession.EffectiveRoles Property

```
property EffectiveRoles: TStringList
```

Available In: Client Applications

Contains a list of the effective roles for the user when the Authenticated property is True.

TServerSession.FullName Property

```
property FullName: String
```

Available In: Client Applications

Indicates the full name for the user when the Authenticated property is True.

TServerSession.KeepAliveResource Property

```
property KeepAliveResource: String
```

Available In: Client Applications

Specifies the resource name that should be used with any keep-alive requests.

TServerSession.ModulesResource Property

```
property ModulesResource: String
```

Available In: Client Applications

Specifies the resource name that should be used with any native server module API requests.

TServerSession.Password Property

```
property Password: String
```

Available In: Client Applications

Specifies the password for the user specified in the UserName property.

TServerSession.RequestExecuting Property

```
property RequestExecuting: Boolean
```

Available In: Client Applications

Indicates whether a server request is currently being executed using the server session.

TServerSession.RequestStatusCode Property

```
property RequestStatusCode: Integer
```

Available In: Client Applications

Indicates the status code of the last server request executed using the server session.

TServerSession.RequestStatusText Property

```
property RequestStatusText: String
```

Available In: Client Applications

Indicates the status text of the last server request executed using the server session.

TServerSession.UserName Property

```
property UserName: String
```

Available In: Client Applications

Specifies the user name to use with the server session.

TServerSession.Authenticate Method

```
procedure Authenticate
```

Available In: Visual Client Applications

Use this method to authenticate the server session using the UserName and Password properties.

TServerSession.CancelRequest Method

```
procedure CancelRequest
```

Available In: Visual Client Applications

Use this method to cancel any web server request that is being currently being executed to satisfy a server session method such as the Authenticate method.

TServerSession.Deauthenticate Method

```
procedure Deauthenticate
```

Available In: Visual Client Applications

Use this method to deauthenticate the server session.

TServerSession.HasPrivilege Method

```
function HasPrivilege(const AName: String): Boolean
```

Available In: Visual Client Applications

Use this method to determine if the authenticated user for the server session has been granted the specified privilege (indirectly through the roles that the user has been granted).

TServerSession.HasRole Method

```
function HasRole(const AName: String): Boolean
```

Available In: Visual Client Applications

Use this method to determine if the authenticated user for the server session has been granted the specified role.

TServerSession.RefreshEffectiveAccess Method

```
procedure RefreshEffectiveAccess
```

Available In: Visual Client Applications

User this method to refresh the EffectivePrivileges and EffectiveRoles properties from the web server. This is useful for situations where the user security has been updated and the server session needs to retrieve the new effective privileges and roles for the authenticated user.

TServerSession.OnAuthenticate Event

```
property OnAuthenticate: TServerSessionAuthenticateEvent
```

Available In: Visual Client Applications

This event is triggered before a server session attempts to perform authentication. You can use an event handler attached to this event to update the UserName and/or Password properties before the authentication occurs.

Set the function result to False to prevent the authentication from proceeding.

> **Note**
> If the function result is set to False, an exception will be raised indicating that the authentication was cancelled.

TServerSession.OnComplete Event

```
property OnComplete: TServerSessionEvent
```

Available In: Visual Client Applications

This event is triggered when a server session request is complete. Use the RequestStatusCode property to determine the status code returned by the web server and, subsequently, whether the session request was successful or not.

TServerSession.OnProgress Event

```
property OnProgress: TServerSessionProgressEvent
```

Available In: Visual Client Applications

This event is triggered as a server session request is executed and represents the current progress of the request.

> **Note**
> The Current and Total parameters passed to the event handler are determined by the browser completing the server request and may differ between browsers.

10.215 TSet Component

Unit: WebCore

Inherits From TObject

Available In: Client and Server Applications

The TSet class represents a set. It is used with the TElement class and descendant classes for determining which aspects of the element has changed during a batch update, but it can be used for any general purpose.

Properties	Methods	Events
Count	Add	OnChanged
Max	All	
	Assign	
	BeginUpdate	
	Copy	
	Create	
	Empty	
	EndUpdate	
	Except	
	Exists	
	Initialize	
	Intersect	
	IsEmpty	
	Range	
	Remove	
	Union	

TSet.Count Property

```
property Count: Integer
```

Available In: Client and Server Applications

Specifies how many items are in the set.

TSet.Max Property

```
property Max: Integer
```

Available In: Client and Server Applications

Specifies the maximum number of items that are, or have been, in the set. This property is useful when you need to iterate over the set and test whether particular values are in the set.

TSet.Add Method

```
function Add(Value: Integer): Boolean
```

Available In: Client and Server Applications

Use this method to add an item to the set.

TSet.All Method

```
procedure All(ACount: Integer)
```

Available In: Client and Server Applications

Use this method to include all items from 0 to ACount-1.

TSet.Assign Method

```
procedure Assign(Value: TSet)
```

Available In: Client and Server Applications

Use this method to assign the contents of a source set to the set.

TSet.BeginUpdate Method

```
procedure BeginUpdate
```

Available In: Client and Server Applications

Use this method to begin a batch update to the set. Batch updates are useful in situations where many changes need to be made to the set, and triggering the OnChanged event on every change would result in performance issues. This method is reference-counted and every time it is called, an internal counter is incremented. Every time the EndUpdate method is called, the counter is decremented. Once the counter reaches zero, the OnChanged event will be triggered.

TSet.Copy Method

```
function Copy: TSet
```

Available In: Client and Server Applications

Use this method to make a new copy of the set and return it as the result.

TSet.Create Method

```
constructor Create(const AValues: TIntegerArray)
```

Available In: Client and Server Applications

Use this method to create a new instance of the TSet class. The AValues parameter is an array of integer values that will be used to initialize the set.

TSet.Empty Method

```
procedure Empty
```

Available In: Client and Server Applications

Use this method to remove all items from the set.

TSet.EndUpdate Method

```
procedure EndUpdate
```

Available In: Client and Server Applications

Use this method to end a batch update to the set. Batch updates are useful in situations where many changes need to be made to the set, and triggering the OnChanged event on every change would result in performance issues. This method is reference-counted and every time it is called, an internal counter is decremented. Every time the BeginUpdate method is called, the counter is incremented. Once the counter reaches zero, the OnChanged event will be triggered.

TSet.Except Method

```
procedure Except(Value: TSet)
```

Available In: Client and Server Applications

Use this method to remove all items from the set that exist in the set passed as the parameter.

TSet.Exists Method

```
function Exists(Value: Integer): Boolean
```

Available In: Client and Server Applications

Use this method to determine if an item exists in the set.

TSet.Initialize Method

```
procedure Initialize(const AValues: TIntegerArray)
```

Available In: Client and Server Applications

Use this method to initialize a set using an array of integer values representing the items that should be in the set.

TSet.Intersect Method

```
procedure Intersect(Value: TSet)
```

Available In: Client and Server Applications

Use this method to include all items from the set that also exist in the set passed as the parameter.

TSet.IsEmpty Method

```
function IsEmpty: Boolean
```

Available In: Client and Server Applications

Use this method to determine if the set is empty.

TSet.Range Method

```
procedure Range(AStart, AEnd: Integer)
```

Available In: Client and Server Applications

Use this method to include a range of items in the set. The AStart parameter indicates the starting item, and the AEnd parameter indicates the ending item.

TSet.Remove Method

```
function Remove(Value: Integer): Boolean
```

Available In: Client and Server Applications

Use this method to remove an item from the set.

TSet.Union Method

```
procedure Union(Value: TSet)
```

Available In: Client and Server Applications

Use this method to add all items in the set passed as the parameter to the current set.

TSet.OnChanged Event

```
property OnChanged: TNotifyEvent
```

Available In: Client and Server Applications

This event is triggered when any member elements are added to the set or removed from the set.

10.216 TShadow Component

Unit: WebUI

Inherits From TElementAttribute

Available In: Visual Client Applications

The TShadow class represents the inset and outset shadows of a UI element or control. Shadows can be a certain color and size, and their placement depends upon whether the shadow is an inset or outset shadow.

Properties	Methods	Events
Blur	SetToDefault	
Color		
HorzOffset		
Spread		
VertOffset		
Visible		

TShadow.Blur Property

```
property Blur: Integer
```

Available In: Visual Client Applications

Specifies the amount of blur for the shadow. The amount of blur is equal to the radius, in pixels, of a circular blur transformation that is applied to the solid shadow. This means that the center point of the blur transformation will always be the edge of the solid shadow before the blur is applied.

TShadow.Color Property

```
property Color: TColor
```

Available In: Visual Client Applications

Specifies the color of the solid shadow before any blur transformation is applied.

TShadow.HorzOffset Property

```
property HorzOffset: Integer
```

Available In: Visual Client Applications

Specifies a horizontal offset, in pixels, to add to the position of the shadow.

TShadow.Spread Property

```
property Spread: Integer
```

Available In: Visual Client Applications

Specifies the amount, in pixels, to increase the size of the solid shadow by before any blur transformation is applied.

TShadow.VertOffset Property

```
property VertOffset: Integer
```

Available In: Visual Client Applications

Specifies a vertical offset, in pixels, to add to the position of the shadow.

TShadow.Visible Property

```
property Visible: Boolean
```

Available In: Visual Client Applications

Specifies whether the shadow is visible.

TShadow.SetToDefault Method

```
procedure SetToDefault
```

Available In: Visual Client Applications

Use this method to reset the shadow's properties to their default values.

10.217 TSizeGrip Component

Unit: WebSizer

Inherits From TSizeGripControl

Available In: Visual Client Applications

The TSizeGrip component represents a size grip control. A size grip control can be used to allow the user to dynamically size the parent control of the size grip, which is always positioned in the bottom right-hand corner of the parent control.

Properties	Methods	Events
Background		OnCaptureEnd
InsetShadow		OnCaptureStart
Opacity		OnCapturing
		OnClick
		OnContextMenu
		OnDblClick
		OnHide
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TSizeGrip.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

TSizeGrip.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Available In: Visual Client Applications

Specifies the inset shadow for the control.

TSizeGrip.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TSizeGrip.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TSizeGrip.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TSizeGrip.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TSizeGrip.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TSizeGrip.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TSizeGrip.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TSizeGrip.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TSizeGrip.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TSizeGrip.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TSizeGrip.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TSizeGrip.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TSizeGrip.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TSizeGrip.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TSizeGrip.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TSizeGrip.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TSizeGrip.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TSizeGrip.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TSizeGrip.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TSizeGrip.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.218 TSizeGripControl Component

Unit: WebSizer

Inherits From TControl

Available In: Visual Client Applications

The TSizeGripControl control is the base class for size grip controls, and contains all of the size grip functionality in the form of public methods and protected properties/events that descendant classes can use to create customized size grip controls.

Properties	Methods	Events

10.219 TSizer Component

Unit: WebSizer

Inherits From TSizerControl

Available In: Visual Client Applications

The TSizer component represents a sizer control. A sizer control can be used to allow the user to dynamically size a specific control in a horizontal or vertical orientation.

Properties	Methods	Events
Border		OnAnimationComplete
Control		OnAnimationsComplete
Corners		OnCaptureEnd
InsetShadow		OnCaptureStart
Opacity		OnCapturing
Orientation		OnClick
Padding		OnContextMenu
		OnDblClick
		OnHide
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TSizer.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TSizer.Control Property

```
property Control: TControl
```

Available In: Visual Client Applications

Specifies the control to be resized.

TSizer.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TSizer.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Available In: Visual Client Applications

Specifies the inset shadow for the control.

TSizer.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TSizer.Orientation Property

```
property Orientation: TSizerOrientation
```

Available In: Visual Client Applications

Specifies the orientation of the sizer control, which determines how the sizer control modifies the dimensions of the control specified in the Control property.

TSizer.Padding Property

```
property Padding: TPadding
```

Available In: Visual Client Applications

Specifies the padding within the client area of the control.

TSizer.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TSizer.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TSizer.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TSizer.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TSizer.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TSizer.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TSizer.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TSizer.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TSizer.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TSizer.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TSizer.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TSizer.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TSizer.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TSizer.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TSizer.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TSizer.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TSizer.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TSizer.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TSizer.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TSizer.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TSizer.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.220 TSizerControl Component

Unit: WebSizer

Inherits From TControl

Available In: Visual Client Applications

The TSizerControl control is the base class for sizer controls, and contains all of the sizer functionality in the form of public methods and protected properties/events that descendant classes can use to create customized sizer controls.

Properties	Methods	Events

10.221 TSlideShow Component

Unit: WebSlide

Inherits From TSlideShowControl

Available In: Visual Client Applications

The TSlideShow component represents a control that can display a slideshow of images with configurable transition and animation effects.

Properties	Methods	Events
CacheSize		OnAnimationComplete
Canvas		OnAnimationsComplete
Cursor		OnCaptureEnd
DisplayTime		OnCaptureStart
FadeTime		OnCapturing
Hint		OnClick
ImageURLs		OnContextMenu
Loop		OnDblClick
Opacity		OnHide
Panning		OnLoadSlide
ZoomFactor		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnRenderSlide
		OnShow
		OnSize
		OnStart
		OnStop
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TSlideShow.CacheSize Property

```
property CacheSize: Integer
```

Available In: Visual Client Applications

Specifies the number of images to cache before the slideshow begins to play. The default value is 2, but you may need to increase this property for low-bandwidth connections in order to avoid issues with playback.

TSlideShow.Canvas Property

```
property Canvas: TCanvasElement
```

Available In: Visual Client Applications

This property provdes access to a TCanvasElement instance that can be used to perform drawing operations on the control from within an OnRenderSlide event handler.

TSlideShow.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TSlideShow.DisplayTime Property

```
property DisplayTime: Integer
```

Available In: Visual Client Applications

Specifies how long, in milliseconds, each slideshow image should be shown before any transition effects are started for the next image. The default value is 8000.

TSlideShow.FadeTime Property

```
property FadeTime: Integer
```

Available In: Visual Client Applications

Specifies how long, in milliseconds, the fade in/fade out effect occurs for each slideshow image transition. The default value is 1000.

TSlideShow.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TSlideShow.ImageURLs Property

```
property ImageURLs: TStrings
```

Available In: Visual Client Applications

Specifies the URLs for all images in the slideshow. You must specify at least CacheSize images or an error will occur when trying to play the slideshow using the Start method.

TSlideShow.Loop Property

```
property Loop: Boolean
```

Available In: Visual Client Applications

Specifies whether the slideshow should restart with the first image after reaching the last image in the ImageURLs property. The default value is False.

TSlideShow.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TSlideShow.Panning Property

```
property Panning: Boolean
```

Available In: Visual Client Applications

Specifies whether the images should be animated so that they pan (move) randomly within the slideshow client area while being displayed. The default value is True.

Note
This property controls one aspect of an effect known as the "Ken Burns Effect", whose name comes from the effects that director Ken Burns made famous with documentaries such as "The Civil War". The ZoomFactor property controls the other aspect, the zooming effect.

TSlideShow.ZoomFactor Property

```
property ZoomFactor: Integer
```

Available In: Visual Client Applications

Specifies whether the images should be animated so that they zoom randomly in or out within the slideshow client area while being displayed. The default value is 15, and valid values are 0 (no zooming) to 100 (very fast zooming).

Note
This property controls one aspect of an effect known as the "Ken Burns Effect", whose name comes from the effects that director Ken Burns made famous with documentaries such as "The Civil War". The Panning property controls the other aspect, the panning effect.

TSlideShow.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TSlideShow.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TSlideShow.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TSlideShow.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TSlideShow.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TSlideShow.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TSlideShow.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TSlideShow.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TSlideShow.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TSlideShow.OnLoadSlide Event

```
property OnLoadSlide: TSlideEvent
```

Available In: Visual Client Applications

This event is triggered whenever a new slide image is loaded. Slide images are only loaded once and then cached until the Stop method is called.

TSlideShow.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TSlideShow.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TSlideShow.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TSlideShow.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TSlideShow.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TSlideShow.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TSlideShow.OnRenderSlide Event

```
property OnRenderSlide: TSlideEvent
```

Available In: Visual Client Applications

This event is triggered whenever a new slide image is rendered. This event is useful for situations where slide images need to be annotated. Because the TSlideShow control is a direct descendant of the TPaint control, it has a Canvas property that can be used to allow for direct drawing from within any event handlers attached to this event.

Note
This event is triggered FramesPerSecond times per second, so it is very important that any code called from within any event handlers for this event is not time-consuming.

TSlideShow.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TSlideShow.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TSlideShow.OnStart Event

```
property OnStart: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the Start method is called, and the slide images being rendering.

Note
This event will not be triggered until CacheSize images are loaded and ready to be displayed.

TSlideShow.OnStop Event

```
property OnStop: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the Stop method is called.

TSlideShow.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TSlideShow.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TSlideShow.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TSlideShow.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

10.222 TSlideShowControl Component

Unit: WebSlide

Inherits From TControl

Available In: Visual Client Applications

The TSlideShowControl control is the base class for slideshow controls, and contains all of the slideshow functionality in the form of public methods and protected properties/events that descendant classes can use to create customized slideshow controls.

Properties	Methods	Events
Started	Start	
	Stop	

TSlideShowControl.Started Property

```
property Started: Boolean
```

Available In: Visual Client Applications

Indicates whether the slideshow has been started using the Start method.

TSlideShowControl.Start Method

```
procedure Start
```

Available In: Visual Client Applications

Use this method to start the slideshow. Once this method is called, the Started property will change to True.

TSlideShowControl.Stop Method

```
procedure Stop
```

Available In: Visual Client Applications

Use this method to stop the slideshow. Once this method is called, the Started property will change to False.

10.223 TStateButtonControl Component

Unit: WebBtns

Inherits From TInputControl

Available In: Visual Client Applications

The TStateButtonControl control is the base class for button controls that represent selection states, and contains all of the selection state functionality in the form of public methods and protected properties/events that descendant classes can use to create customized selection state button controls.

Properties	Methods	Events

10.224 TStream Component

Unit: WebSrvr

Inherits From TExternalObject

Available In: Server Applications

The TStream class is the base class used to represent a binary stream of data, and is used by the TMemoryStream and TFileStream classes to implement stream access to blocks of memory and files, as well as used by the TDataColumn class to provide stream access to BLOBs.

Properties	Methods	Events
AvailableSize	Compress	
MaximumSize	ComputeCRC32	
Position	ComputeHash	
Size	CopyFrom	
	Decompress	
	Decrypt	
	Encrypt	
	LoadFromFile	
	LoadFromStream	
	ReadBinaryChars	
	ReadBoolean	
	ReadChar	
	ReadChars	
	ReadDateTime	
	ReadFloat64	
	ReadInt16	
	ReadInt32	
	ReadInt64	
	ReadInt8	
	ReadString	
	ReadUInt16	
	ReadUInt32	
	ReadUInt64	
	ReadUInt8	
	ReadUTF8Chars	
	ReadUTF8String	
	SaveToFile	

	SaveToStream	
	Seek	
	WriteBinaryChars	
	WriteBoolean	
	WriteChar	
	WriteChars	
	WriteDateTime	
	WriteFloat64	
	WriteInt16	
	WriteInt32	
	WriteInt64	
	WriteInt8	
	WriteString	
	WriteUInt16	
	WriteUInt32	
	WriteUInt64	
	WriteUInt8	
	WriteUTF8Chars	
	WriteUTF8String	

TStream.AvailableSize Property

```
property AvailableSize: Integer
```

Available In: Server Applications

Indicates the available size from the current position up to the size of the stream.

TStream.MaximumSize Property

```
property MaximumSize: Integer
```

Available In: Server Applications

Indicates the maximum allowed size for the stream. Different types of streams have different maximum sizes, but the default maximum size is Max(<Native Pointer Size>). File streams will have much higher maximum sizes that depend upon the file system in use.

TStream.Position Property

```
property Position: Integer
```

Available In: Server Applications

Specifies the current position of the stream pointer in the stream. Every stream contains a current position that is updated as various stream methods are used to read/write to/from the stream.

Note
For memory streams, specifying an invalid position in the stream will cause an exception to be raised.

TStream.Size Property

```
property Size: Integer
```

Available In: Server Applications

Specifies the size of the stream. If an assigned value is larger than the current stream size and the stream can be dynamically expanded, then the stream will size will be increased accordingly. If an assigned value is smaller than the current stream size and the stream can be dynamically contracted, then the stream will be truncated and the stream's current position moved, if necessary.

TStream.Compress Method

```
procedure Compress(DestStream: TStream; CompressionFormat:
      TCompressionFormat=cfZLib; CompressionLevel:
      Integer=DEFAULT_COMPRESSION)
```

Available In: Server Applications

Use this method to compress the data in the stream into the destination stream using the specified compression algorithm. The compression starts from the stream's Position property and the stream's AvailableSize property is used as the count. The compressed data is written to the destination stream starting at the destination stream's current position.

TStream.ComputeCRC32 Method

```
function ComputeCRC32(const Count: Integer=0): Integer
```

Available In: Server Applications

Use this method to calculate the cyclic redundancy check (CRC) for the stream using the CRC-32 algorithm. The CRC is computed from the stream's current position for the specified count. If the count is 0, then the stream's AvailableSize property is used as the count.

TStream.ComputeHash Method

```
function ComputeHash(AHashType: THashType; const Count:
      Integer=0): String
```

Available In: Server Applications

Use this method to compute the cryptographic hash for the stream using the specified hash algorithm. The hash is computed from the stream's current position for the specified count. If the count is 0, then the stream's AvailableSize property is used as the count.

TStream.CopyFrom Method

```
function CopyFrom(Stream: TStream; const Count: Integer=0):
      Integer
```

Available In: Server Applications

Use this method to copy the specified number of bytes from a source stream into the stream. The copy is performed from the source stream's current position for the specified count. If the count is 0, then the source stream's AvailableSize property is used as the count.

TStream.Decompress Method

```
procedure Decompress(DestStream: TStream; CompressionFormat:
      TCompressionFormat=cfZLib)
```

Available In: Server Applications

Use this method to decompress the data in the stream into the destination stream using the specified decompression algorithm. The decompression starts from the stream's current position and the stream's AvailableSize property is used as the count. The decompressed data is written to the destination stream starting at the destination stream's current position.

TStream.Decrypt Method

```
procedure Decrypt(const Password: String; DestStream: TStream;
      EncryptionType: TEncryptionType=etAES128; const Count:
      Integer=0)
```

Available In: Server Applications

Use this method to decrypt the data in the stream into the destination stream using the specified encryption algorithm. The decryption starts from the stream's current position for the specified count. If the count is 0, then the stream's AvailableSize property is used as the count. The decrypted data is written to the destination stream starting at the destination stream's current position.

TStream.Encrypt Method

```
procedure Encrypt(const Password: String; DestStream: TStream;
      EncryptionType: TEncryptionType=etAES128; const Count:
      Integer=0)
```

Available In: Server Applications

Use this method to encrypt the data in the stream into the destination stream using the specified encryption algorithm. The encryption starts from the stream's current position for the specified count. If the count is 0, then the stream's AvailableSize property is used as the count. The encrypted data is written to the destination stream starting at the destination stream's current position.

TStream.LoadFromFile Method

```
procedure LoadFromFile(const FileName: String)
```

Available In: Server Applications

Use this method to write all of the data from the specified file into the stream. Before the data is written, the stream's Size property is set to 0.

TStream.LoadFromStream Method

```
procedure LoadFromStream(Stream: TStream)
```

Available In: Server Applications

Use this method to write all of the data from the specified stream into the stream. Before the data is written, the stream's Size property is set to 0 and the source stream's Position property is set to 0.

TStream.ReadBinaryChars Method

```
function ReadBinaryChars(const Count: Integer): String
```

Available In: Server Applications

Use this method to read the binary data in the stream as a hexidecimal-encoded UTF-16 string. The specified length determines how many bytes are read from the stream's current position. Hexidecimal-encoded binary data can be written to the stream using the WriteBinaryChars method.

> **Note**
> If the data cannot be read for any reason, an exception will be raised.

TStream.ReadBoolean Method

```
function ReadBoolean: Boolean
```

Available In: Server Applications

Use this method to read a boolean (unsigned, 8-bit integer) True (1) or False (0) value from the stream, starting at the stream's current position.

> **Note**
> If the value cannot be read for any reason, an exception will be raised.

TStream.ReadChar Method

```
function ReadChar: Char
```

Available In: Server Applications

Use this method to read a 16-bit Unicode character from the stream's current position.

Note
If the value cannot be read for any reason, an exception will be raised.

TStream.ReadChars Method

```
function ReadChars(const Count: Integer): String
```

Available In: Server Applications

Use this method to read the UTF-16-encoded string with the specified length from the stream's current position. Such strings are written to the stream using the WriteChars method.

> **Note**
> If the characters cannot be read for any reason, an exception will be raised.

TStream.ReadDateTime Method

```
function ReadDateTime: Integer
```

Available In: Server Applications

Use this method to read a date-time (64-bit, signed integer) value from the stream's current position.

> **Note**
> If the value cannot be read for any reason, an exception will be raised.

TStream.ReadFloat64 Method

```
function ReadFloat64: Double
```

Available In: Server Applications

Use this method to read an 64-bit, double-precision floating-point value from the stream's current position.

Note
If the value cannot be read for any reason, an exception will be raised.

TStream.ReadInt16 Method

```
function ReadInt16: Integer
```

Available In: Server Applications

Use this method to read a 16-bit signed integer value from the stream's current position.

> **Note**
> If the value cannot be read for any reason, an exception will be raised.

TStream.ReadInt32 Method

```
function ReadInt32: Integer
```

Available In: Server Applications

Use this method to read a 32-bit signed integer value from the stream's current position.

Note
If the value cannot be read for any reason, an exception will be raised.

TStream.ReadInt64 Method

```
function ReadInt64: Integer
```

Available In: Server Applications

Use this method to read a 64-bit signed integer value from the stream's current position.

Note
If the value cannot be read for any reason, an exception will be raised.

TStream.ReadInt8 Method

```
function ReadInt8: Integer
```

Available In: Server Applications

Use this method to read an 8-bit signed integer value from the stream's current position.

Note
If the value cannot be read for any reason, an exception will be raised.

TStream.ReadString Method

```
function ReadString: String
```

Available In: Server Applications

Use this method to read a UTF-16-encoded string from the stream's current position. Strings are written to the stream with a 32-bit, signed integer length-prefix when using the WriteString method. This method reads the string using the length prefix.

Note
If the string cannot be read for any reason, an exception will be raised.

TStream.ReadUInt16 Method

```
function ReadUInt16: Integer
```

Available In: Server Applications

Use this method to read a 16-bit unsigned integer value from the stream's current position.

Note
If the value cannot be read for any reason, an exception will be raised.

TStream.ReadUInt32 Method

```
function ReadUInt32: Integer
```

Available In: Server Applications

Use this method to read a 32-bit unsigned integer value from the stream's current position.

> **Note**
> If the value cannot be read for any reason, an exception will be raised.

TStream.ReadUInt64 Method

```
function ReadUInt64: Integer
```

Available In: Server Applications

Use this method to read a 64-bit unsigned integer value from the stream's current position.

> **Note**
> If the value cannot be read for any reason, an exception will be raised.

TStream.ReadUInt8 Method

```
function ReadUInt8: Integer
```

Available In: Server Applications

Use this method to read an 8-bit unsigned integer value from the stream's current position.

Note
If the value cannot be read for any reason, an exception will be raised.

TStream.ReadUTF8Chars Method

```
function ReadUTF8Chars(const Count: Integer): String
```

Available In: Server Applications

Use this method to read a UTF-8-encoded string with the specified length from the stream's current position. Such strings are written to the stream using the WriteUTF8Chars method.

> **Note**
> If the characters cannot be read for any reason, an exception will be raised.

TStream.ReadUTF8String Method

```
function ReadUTF8String: String
```

Available In: Server Applications

Use this method to read a UTF8-encoded string from the stream's current position. Such strings are written to the stream with a 32-bit, signed integer length-prefix when using the WriteUTF8String method. This method reads the string using the length prefix.

Note
If the string cannot be read for any reason, an exception will be raised.

TStream.SaveToFile Method

```
procedure SaveToFile(const FileName: String)
```

Available In: Server Applications

Use this method to write all of the data from the stream into the specified file. If the file already exists, it will be overwritten. Before the data is written, the stream's Position property is set to 0.

TStream.SaveToStream Method

```
procedure SaveToStream(Stream: TStream)
```

Available In: Server Applications

Use this method to write all of the data from the stream into the destination stream. Before the data is written, the destination stream's Size property is set to 0 and the stream's Position property is set to 0.

TStream.Seek Method

```
function Seek(const Offset: Integer; Origin: TStreamOrigin):
      Integer
```

Available In: Server Applications

Use this method to move the stream's current position to the specified offset, relative to the specified origin.

Note
For memory streams, specifying parameters that result in an invalid position in the stream will cause an exception to be raised.

TStream.WriteBinaryChars Method

```
procedure WriteBinaryChars(const Value: String)
```

Available In: Server Applications

Use this method to write binary data in a hexadecimal-encoded string to the stream's current position. If the data contains one or more invalid hexadecimal characters ("0"-"9" and "A"-"F"), an exception will be raised.

Note
If the value cannot be written for any reason, an exception will be raised. However, most types of streams can be expanded dynamically, so writing past the end of the stream will simply result in an increase in the stream size.

TStream.WriteBoolean Method

```
procedure WriteBoolean(Value: Boolean)
```

Available In: Server Applications

Use this method to write a boolean (unsigned, 8-bit integer) True (1) or False (0) value to the stream, starting at the stream's current position.

Note
If the value cannot be written for any reason, an exception will be raised. However, most types of streams can be expanded dynamically, so writing past the end of the stream will simply result in an increase in the stream size.

TStream.WriteChar Method

```
procedure WriteChar(Value: Char)
```

Available In: Server Applications

Use this method to write a 16-bit Unicode character to the stream's current position.

> **Note**
> If the value cannot be written for any reason, an exception will be raised. However, most types of streams can be expanded dynamically, so writing past the end of the stream will simply result in an increase in the stream size.

TStream.WriteChars Method

```
procedure WriteChars(const Value: String)
```

Available In: Server Applications

Use this method to write a UTF-16-encoded string to the stream's current position.

> **Note**
> If the value cannot be written for any reason, an exception will be raised. However, most types of streams can be expanded dynamically, so writing past the end of the stream will simply result in an increase in the stream size.

TStream.WriteDateTime Method

```
procedure WriteDateTime(const Value: Integer)
```

Available In: Server Applications

Use this method to write a date-time (64-bit, signed integer) value to the stream's current position.

> **Note**
> If the value cannot be written for any reason, an exception will be raised. However, most types of streams can be expanded dynamically, so writing past the end of the stream will simply result in an increase in the stream size.

TStream.WriteFloat64 Method

```
procedure WriteFloat64(const Value: Double)
```

Available In: Server Applications

Use this method to write an 64-bit, double-precision floating-point value to the stream's current position.

Note
If the value cannot be written for any reason, an exception will be raised. However, most types of streams can be expanded dynamically, so writing past the end of the stream will simply result in an increase in the stream size.

TStream.WriteInt16 Method

```
procedure WriteInt16(Value: Integer)
```

Available In: Server Applications

Use this method to write a 16-bit signed integer value to the stream's current position.

> **Note**
> If the value cannot be written for any reason, an exception will be raised. However, most types of streams can be expanded dynamically, so writing past the end of the stream will simply result in an increase in the stream size.

TStream.WriteInt32 Method

```
procedure WriteInt32(Value: Integer)
```

Available In: Server Applications

Use this method to write a 32-bit signed integer value to the stream's current position.

Note
If the value cannot be written for any reason, an exception will be raised. However, most types of streams can be expanded dynamically, so writing past the end of the stream will simply result in an increase in the stream size.

TStream.WriteInt64 Method

```
procedure WriteInt64(const Value: Integer)
```

Available In: Server Applications

Use this method to write a 64-bit signed integer value to the stream's current position.

Note
If the value cannot be written for any reason, an exception will be raised. However, most types of streams can be expanded dynamically, so writing past the end of the stream will simply result in an increase in the stream size.

TStream.WriteInt8 Method

```
procedure WriteInt8(Value: Integer)
```

Available In: Server Applications

Use this method to write an 8-bit signed integer value to the stream's current position.

Note
If the value cannot be written for any reason, an exception will be raised. However, most types of streams can be expanded dynamically, so writing past the end of the stream will simply result in an increase in the stream size.

TStream.WriteString Method

```
procedure WriteString(const Value: String)
```

Available In: Server Applications

Use this method to write a UTF-16-encoded string to the stream's current position. The string is written to the stream with a 32-bit, signed integer length-prefix.

Note
If the value cannot be written for any reason, an exception will be raised. However, most types of streams can be expanded dynamically, so writing past the end of the stream will simply result in an increase in the stream size.

TStream.WriteUInt16 Method

```
procedure WriteUInt16(Value: Integer)
```

Available In: Server Applications

Use this method to write a 16-bit unsigned integer value to the stream's current position.

Note
If the value cannot be written for any reason, an exception will be raised. However, most types of streams can be expanded dynamically, so writing past the end of the stream will simply result in an increase in the stream size.

TStream.WriteUInt32 Method

```
procedure WriteUInt32(Value: Integer)
```

Available In: Server Applications

Use this method to write a 32-bit unsigned integer value to the stream's current position.

Note
If the value cannot be written for any reason, an exception will be raised. However, most types of streams can be expanded dynamically, so writing past the end of the stream will simply result in an increase in the stream size.

TStream.WriteUInt64 Method

```
procedure WriteUInt64(const Value: Integer)
```

Available In: Server Applications

Use this method to write a 64-bit unsigned integer value to the stream's current position.

> **Note**
> If the value cannot be written for any reason, an exception will be raised. However, most types of streams can be expanded dynamically, so writing past the end of the stream will simply result in an increase in the stream size.

TStream.WriteUInt8 Method

```
procedure WriteUInt8(Value: Integer)
```

Available In: Server Applications

Use this method to write an 8-bit unsigned integer value to the stream's current position.

> **Note**
> If the value cannot be written for any reason, an exception will be raised. However, most types of streams can be expanded dynamically, so writing past the end of the stream will simply result in an increase in the stream size.

TStream.WriteUTF8Chars Method

```
procedure WriteUTF8Chars(const Value: String)
```

Available In: Server Applications

Use this method to write a UTF-8-encoded string to the stream's current position.

Note
If the value cannot be written for any reason, an exception will be raised. However, most types of streams can be expanded dynamically, so writing past the end of the stream will simply result in an increase in the stream size.

TStream.WriteUTF8String Method

```
procedure WriteUTF8String(const Value: String)
```

Available In: Server Applications

Use this method to write a UTF-8-encoded string to the stream's current position. The string is written to the stream with a 32-bit, signed integer length-prefix.

> **Note**
> If the value cannot be written for any reason, an exception will be raised. However, most types of streams can be expanded dynamically, so writing past the end of the stream will simply result in an increase in the stream size.

10.225 TStreetViewControlOptions Component

Unit: WebMaps

Inherits From TMapOption

Available In: Visual Client Applications

The TStreetViewControlOptions class controls how the street view control is configured in a TMap control. These street view control options correspond to the street view control options available for maps in the Google Maps API.

Properties	Methods	Events
Position		

TStreetViewControlOptions.Position Property

```
property Position: TMapControlPosition
```

Available In: Visual Client Applications

Specifies the position of the street view control.

10.226 TStringBuilder Component

Unit: WebCore

Inherits From TObject

Available In: Client and Server Applications

The TStringBuilder class is used to manipulate individual characters in a string. Strings are immutable, meaning that you cannot insert, modify, or delete characters in-place within the string. The TStringBuilder class allows for such operations on a string.

Properties	Methods	Events
Chars	Append	
Length	Create	
	GetString	
	Insert	
	Remove	
	ToString	

TStringBuilder.Chars Property

```
property Chars[Index: Integer]: Char
```

Available In: Client and Server Applications

Use this property to get access to the individual characters in the string as an array.

TStringBuilder.Length Property

```
property Length: Integer
```

Available In: Client and Server Applications

Specifies the length of the string.

TStringBuilder.Append Method

```
procedure Append(const Value: String)

procedure Append(Value: Char)
```

Available In: Client and Server Applications

Use this method to append a single character, or another string, to the string.

TStringBuilder.Create Method

```
constructor Create(const Value: String='')
```

Available In: Client and Server Applications

Use this method to create a new instance of the TStringBuilder class. The optional Value parameter is used to initialize the string builder instance with a specific string value.

TStringBuilder.GetString Method

```
function GetString(Index: Integer; Count: Integer): String
```

Available In: Client and Server Applications

Use this method to retrieve a string containing the specified count of characters from the specified index in the string. If the index plus the count of characters is greater than the total number of characters in the string, then the length of the resultant string will be less than the count that is specified.

TStringBuilder.Insert Method

```
procedure Insert(Position: Integer; const Value: String)

procedure Insert(Position: Integer; Value: Char)
```

Available In: Client and Server Applications

Use this method to insert a single character, or another string, into the string at a specific position.

TStringBuilder.Remove Method

```
procedure Remove(Position: Integer; Count: Integer)
```

Available In: Client and Server Applications

Use this method to remove one or more characters from the string.

TStringBuilder.ToString Method

```
function ToString: String
```

Available In: Client and Server Applications

Use this method to retrieve the actual string value after you are finished manipulating the string.

10.227 TStringList Component

Unit: WebCore

Inherits From TStrings

Available In: Client and Server Applications

The TStringList class implements the ancestor TStrings abstract class by providing string storage and sorting/searching functionality.

Properties	Methods	Events
SortCaseInsensitive	Find	
Sorted	Sort	
SortLocaleInsensitive		

TStringList.SortCaseInsensitive Property

```
property SortCaseInsensitive: Boolean
```

Available In: Client and Server Applications

Specifies that the list of strings should be sorted in a case-insensitive fashion. The SortCaseInsensitive and SortLocaleInsensitive properties determine how the strings are compared during the sort:

Properties	Function Used
SortCaseInsensitive=False SortLocaleInsensitive=True	CompareStr
SortCaseInsensitive=True SortLocaleInsensitive=True	CompareText
SortCaseInsensitive=False SortLocaleInsensitive=False	LocaleCompareStr
SortCaseInsensitive=True SortLocaleInsensitive=False	LocaleCompareText

TStringList.Sorted Property

```
property Sorted: Boolean
```

Available In: Client and Server Applications

Specifies that the list of strings is sorted. When the list of strings is sorted, any operations that modify the list automatically trigger a re-sort of the strings in the list.

TStringList.SortLocaleInsensitive Property

```
property SortLocaleInsensitive: Boolean
```

Available In: Client and Server Applications

Specifies that the list of strings should be sorted in a locale-insensitive fashion. The SortCaseInsensitive and SortLocaleInsensitive properties determine how the strings are compared during the sort:

Properties	Function Used
SortCaseInsensitive=False SortLocaleInsensitive=True	CompareStr
SortCaseInsensitive=True SortLocaleInsensitive=True	CompareText
SortCaseInsensitive=False SortLocaleInsensitive=False	LocaleCompareStr
SortCaseInsensitive=True SortLocaleInsensitive=False	LocaleCompareText

TStringList.Find Method

```
function Find(const Value: String; NearestMatch: Boolean=False):
      Integer
```

Available In: Client and Server Applications

Use this method to perform a binary search of the list of strings. The Sorted property must be True or calling this method will result in an exception being raised. The SortCaseInsensitive and SortLocaleInsensitive properties determine how the strings are compared during the search:

Properties	Function Used
SortCaseInsensitive=False SortLocaleInsensitive=True	CompareStr
SortCaseInsensitive=True SortLocaleInsensitive=True	CompareText
SortCaseInsensitive=False SortLocaleInsensitive=False	LocaleCompareStr
SortCaseInsensitive=True SortLocaleInsensitive=False	LocaleCompareText

TStringList.Sort Method

```
procedure Sort
```

Available In: Client and Server Applications

Use this method to sort the list of strings manually. Normally, the Sorted property would be used to keep the list sorted at all times, but sometimes the developer needs a finer level of control over when the list is sorted. The SortCaseInsensitive and SortLocaleInsensitive properties determine how the strings are compared during the sort:

Properties	Function Used
SortCaseInsensitive=False SortLocaleInsensitive=True	CompareStr
SortCaseInsensitive=True SortLocaleInsensitive=True	CompareText
SortCaseInsensitive=False SortLocaleInsensitive=False	LocaleCompareStr
SortCaseInsensitive=True SortLocaleInsensitive=False	LocaleCompareText

10.228 TStrings Component

Unit: WebCore

Inherits From TAbstractList

Available In: Client and Server Applications

The TStrings class is an abstract class that is used to manage a list of strings. Because it is an abstract class, the storage of the strings is not implemented in this class, but is rather left to descendant classes to implement. For example, the TStringList descendant class actually contains an internal array that is used to store the list of strings.

Properties	Methods	Events
Count	Add	
LineSeparator	Clear	
Names	Delete	
NameValueSeparator	IndexOf	
Strings	IndexOfName	
Text	IndexOfValue	
ValueFromIndex	Insert	
Values	Remove	

TStrings.Count Property

```
property Count: Integer
```

Available In: Client and Server Applications

Indicates the total number of strings in the list.

TStrings.LineSeparator Property

```
property LineSeparator: String
```

Available In: Client and Server Applications

Specifies the character, or characters, to use to separate multiple strings into the list of strings when assigning a string value to the Text property. These characters are also used to format a string when reading the Text property. The default value for this property is the carriage return (0x0D) character and linefeed (0x0A) character.

Note
In the JavaScript runtime environment, the linefeed character is sufficient to express a line break, so controls like the TMultiLineEdit component use a LineSeparator of just a linefeed character for its Lines property.

TStrings.Names Property

```
property Names[Index: Integer]: String
```

Available In: Client and Server Applications

Allows access to the name portion of name/value pairs in the list of strings. The index specifies the position of the name in the list that you wish to access or assign.

TStrings.NameValueSeparator Property

```
property NameValueSeparator: Char
```

Available In: Client and Server Applications

Specifies the character, or characters, to use to separate name/value pairs in the list of strings. The Names, ValueFromIndex, and Values properties can be used to work wtih the name/value pairs. The default value for this property is the equals (=) character.

TStrings.Strings Property

```
property Strings[Index: Integer]: String
```

Available In: Client and Server Applications

Allows indexed access to all strings in the list.

TStrings.Text Property

```
property Text: String
```

Available In: Client and Server Applications

Specifies the list of strings as one formatted string. The LineSeparator property is used to format the strings in the list into one string.

TStrings.ValueFromIndex Property

```
property ValueFromIndex[Index: Integer]: String
```

Available In: Client and Server Applications

Allows access to the value portion of name/value pairs in the list of strings. The index specifies the position of the value in the list that you wish to access or assign.

TStrings.Values Property

```
property Values[const Name: String]: String
```

Available In: Client and Server Applications

Allows access to the value portion of name/value pairs in the list of strings. The name specifies the name portion of the name/value pair in the list that you wish to access or assign a value for.

TStrings.Add Method

```
function Add(const Value: String): Integer
```

Available In: Client and Server Applications

Use this method to add a string to the end of the list of strings. The string will be added to the end of the list, and the index of the string in the list will be returned.

TStrings.Clear Method

```
procedure Clear
```

Available In: Client and Server Applications

Use this method to remove all strings from the list.

TStrings.Delete Method

```
procedure Delete(Index: Integer)
```

Available In: Client and Server Applications

Use this method to remove a string from the list of strings using its index.

TStrings.IndexOf Method

```
function IndexOf(const Value: String; StartIndex: Integer=0;
      PartialMatch: Boolean=False): Integer
```

Available In: Client and Server Applications

Use this method to return the index of a particular string in the list of strings. The StartIndex parameter indicates the index into the list of strings at which to start the search, and the PartialMatch parameter indicates whether only the length of the search string should be used when performing the comparisons. The SameText function is used by this method to compare the strings.

TStrings.IndexOfName Method

```
function IndexOfName(const Name: String; StartIndex: Integer=0;
      PartialMatch: Boolean=False): Integer
```

Available In: Client and Server Applications

Use this method to return the index of a particular name portion of the name/value pairs in the list of strings. The SameText function is used by this method to compare the names.

TStrings.IndexOfValue Method

```
function IndexOfValue(const Value: String): Integer
```

Available In: Client and Server Applications

Use this method to return the index of a particular value portion of the name/value pairs in the list of strings. The SameText function is used by this method to compare the values.

TStrings.Insert Method

```
procedure Insert(Index: Integer; const Value: String)
```

Available In: Client and Server Applications

Use this method to insert a string at a specific index in the list of strings.

TStrings.Remove Method

```
function Remove(const Value: String): Integer
```

Available In: Client and Server Applications

Use this method to remove a string from the list of strings by its value. The IndexOf method is used by this method to find the string.

10.229 TSurface Component

Unit: WebForms

Inherits From TScrollableControl

Available In: Visual Client Applications

The TSurface component represents the application surface that is included with every visual application and provides properties and methods for iterating through all forms in the application, specifying the layout of the application surface, and showing message and progress dialogs.

Properties	Methods	Events
ActiveForm	EndModal	OnClientSize
Background	HideProgress	OnScrollSize
Cursor	MessageDlg	OnSize
MaxDialogWidth	ShowMessage	
ModalOverlay	ShowProgress	
ScrollBars	StartModal	
ScrollSupport		

TSurface.ActiveForm Property

```
property ActiveForm: TFormControl
```

Available In: Visual Client Applications

Indicates the active TFormControl instance in the application.

TSurface.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background for the control.

TSurface.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TSurface.MaxDialogWidth Property

```
property MaxDialogWidth: Integer
```

Available In: Visual Client Applications

Specifies the maximum dialog width to use for system-generated message dialogs. The default value is 50% of the available width of the application viewport.

With mobile applications, it may be desirable to adjust this property to allow for full-screen message dialogs that work better with narrow portrait orientations.

TSurface.ModalOverlay Property

```
property ModalOverlay: TModalOverlay
```

Available In: Visual Client Applications

Provides access to the modal overlay instance for the application's surface.

TSurface.ScrollBars Property

```
property ScrollBars: TScrollBars
```

Available In: Visual Client Applications

Specifies which scrollbars to show, if any.

Note
Even if this property is set to sbHorizontal, sbVertical, or sbBoth, a scrollbar will only be shown if the size of the contents and/or the child controls of the control exceed the client rectangle for the control.

TSurface.ScrollSupport Property

```
property ScrollSupport: TScrollSupport
```

Available In: Visual Client Applications

Specifies the directions in which the control can be scrolled, if any.

> **Note**
> This property only applies to scrolling via touch and mouse wheel movements, and does not apply to programmatic scrolling or scrolling via the scroll bars themselves.

TSurface.EndModal Method

```
procedure EndModal
```

Available In: Visual Client Applications

Use this method to decrement the global modal reference count, and if the reference count is 0, end the modal state of the application. The StartModal method puts the user interface into a modal state and increments the global modal reference count.

TSurface.HideProgress Method

```
procedure HideProgress
```

Available In: Visual Client Applications

Use this method to decrement the global progress reference count, and if the reference count is 0, hide the active progress dialog. The ShowProgress method shows a progress dialog and increments the progress reference count.

TSurface.MessageDlg Method

```
procedure MessageDlg(const Msg: String; const DlgCaption: String;
       DlgType: TMsgDlgType; const Buttons: TMsgDlgBtns;
      DefaultButton: TMsgDlgBtn; MsgDlgResult: TMsgDlgResultEvent=nil;
      DlgClose: Boolean=False; AnimationStyle: TAnimationStyle=asNone;
      AnimationDuration: Integer=0)
```

Available In: Visual Client Applications

Use this method to display a modal message dialog. Please see the MessageDlg procedure for more information on the parameters to this method.

TSurface.ShowMessage Method

```
procedure ShowMessage(const Msg: String; const DlgCaption:
      String=''; AnimationStyle: TAnimationStyle=asNone;
      AnimationDuration: Integer=0)
```

Available In: Visual Client Applications

Use this method to show a simple modal message dialog. The Msg parameter indicates the message to show.

TSurface.ShowProgress Method

```
procedure ShowProgress(const Msg: String; AnimationStyle:
      TAnimationStyle=asNone; AnimationDuration: Integer=0)
```

Available In: Visual Client Applications

Use this method to show a modal progress dialog and increment the global progress reference count. The HideProgress method decrements the reference count and hides any active progress dialog.

TSurface.StartModal Method

```
procedure StartModal
```

Available In: Visual Client Applications

Use this method to put the user interface into a modal state and increment the global modal reference count. The EndModal method decrements the reference count and takes the application out of the modal state when the reference count reaches zero.

When the user interface is in the modal state, the application user cannot use the mouse/touch or keyboard to interact with the user interface. This is useful in situations where you want to execute an asynchronous operation like a server request and don't want to allow the user to interact with the user interface while the server request is in progress.

TSurface.OnClientSize Event

```
property OnClientSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the width and/or height of the client area of the control is changed.

TSurface.OnScrollSize Event

```
property OnScrollSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the width and/or height of the scrollable area of the control is changed.

TSurface.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

10.230 TTab Component

Unit: WebPages

Inherits From TControl

Available In: Visual Client Applications

The TTab component represents the tab for a TPage instance within a TPagePanel control. Each page instance contains a reference to a tab control via its Tab property. The properties in the tab can be modified to affect the tab caption and how the tab is sized and formatted.

Properties	Methods	Events
AllowClose		
AutoWidth		
Caption		
Font		
Hint		

TTab.AllowClose Property

```
property AllowClose: Boolean
```

Available In: Visual Client Applications

Specifies whether the close button should be shown in the tab.

TTab.AutoWidth Property

```
property AutoWidth: Boolean
```

Available In: Visual Client Applications

Specifies whether the width of the tab should be automatically set based upon the Caption and Font properties.

TTab.Caption Property

```
property Caption: String
```

Available In: Visual Client Applications

Specifies the caption to display on the tab.

TTab.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TTab.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

10.231 TTextAreaElement Component

Unit: WebUI

Inherits From TInputElement

Available In: Visual Client Applications

The TTextAreaElement class is the base element class for text area elements, and contains all of the text area functionality in the form of public methods and properties/events that control classes can use to create text area controls.

> **Note**
> This element does not provide support for text area elements at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events

10.232 TTextInputElement Component

Unit: WebUI

Inherits From TInputElement

Available In: Visual Client Applications

The TTextInputElement class is the base element class for text input elements, and contains all of the text input functionality in the form of public methods and properties/events that control classes can use to create text input controls.

> **Note**
> This element does not provide support for text input elements at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events
InputType		

TTextInputElement.InputType Property

```
property InputType: TTextInputType
```

Available In: Visual Client Applications

Specifies the type of text being input into the element by the user. This information is used by the browser to determine how to display and edit the text. For example, in touch environments, this property is used to determine which soft keyboard to display to the user.

10.233 TTimer Component

Unit: WebComps

Inherits From TComponent

Available In: Client Applications

The TTimer component represents a timer that triggers an OnTimer event whenever the interval, specified in milliseconds, elapses. Timers are asynchronous, meaning that they can trigger the OnTimer event even while the user is performing other tasks in the web browser.

Properties	Methods	Events
Enabled		OnTimer
Interval		

TTimer.Enabled Property

```
property Enabled: Boolean
```

Available In: Client Applications

Specifies whether the timer is enabled. Whenever the timer is enabled, the Interval for the timer starts from zero. The default value is True.

TTimer.Interval Property

```
property Interval: Integer
```

Available In: Client Applications

Specifies the interval for the timer in milliseconds. Whenever the interval elapses, the OnTimer event is triggered. The default value is 1000 milliseconds.

TTimer.OnTimer Event

```
property OnTimer: TNotifyEvent
```

Available In: Client Applications

This event is triggered whenever the Interval specified for the timer elapses.

10.234 TToolBar Component

Unit: WebTlbrs

Inherits From TToolBarControl

Available In: Visual Client Applications

The TToolBar class represents a toolbar control. A toolbar control contains 0 or more TToolBarButton instances that serve as non-focusable buttons, and is ideal for menus.

Properties	Methods	Events
Background		OnAnimationComplete
Border		OnAnimationsComplete
Corners		OnButtonClick
Cursor		OnHide
Hint		OnMove
InsetShadow		OnShow
MultiSelect		OnSize
Opacity		

TToolBar.Background Property

```
property Background: TBackground
```

Available In: Visual Client Applications

Specifies the background of the control.

TToolBar.Border Property

```
property Border: TBorder
```

Available In: Visual Client Applications

Specifies the border for the control.

TToolBar.Corners Property

```
property Corners: TCorners
```

Available In: Visual Client Applications

Specifies the horizontal and vertical radii for the corners of the control.

TToolBar.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TToolBar.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TToolBar.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Available In: Visual Client Applications

Specifies the inset shadow for the control.

TToolBar.MultiSelect Property

```
property MultiSelect: Boolean
```

Available In: Visual Client Applications

Specifies whether multiple toolbar buttons can have their Selected property set to True at the same time. The default value is True.

Note
Toolbar buttons can only be selected if their AllowSelection property is set to True.

TToolBar.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TToolBar.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TToolBar.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TToolBar.OnButtonClick Event

```
property OnButtonClick: TClickEvent
```

Available In: Visual Client Applications

This event is triggered whenever one of the toolbar buttons is clicked.

Return True to allow the default click behavior and False to prevent the default click behavior from occurring.

TToolBar.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TToolBar.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TToolBar.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TToolBar.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

10.235 TToolBarButton Component

Unit: WebTlbrs

Inherits From TRepeatControl

Available In: Visual Client Applications

The TToolBarButton class represents a button on a TToolBar control. Toolbar buttons consist of an icon and an optional caption, and cannot obtain focus.

Properties	Methods	Events
AllowSelection		OnClick
AutoWidth		OnHide
Caption		OnShow
Cursor		
Enabled		
Font		
Hint		
Icon		
Index		
ParentToolBar		
RepeatClick		
RepeatClickInterval		
Selected		

TToolBarButton.AllowSelection Property

```
property AllowSelection: Boolean
```

Available In: Visual Client Applications

Specifies whether the toolbar button is selectable. If a toolbar button is selectable, then its Selected property can be modified to toggle the button to and from a "pushed" state.

Note
By default, multiple toolbar buttons within the same toolbar can have their Selected property set to True at the same time. This behavior is controlled by the TToolBar MultiSelect property. If the MultiSelect property is set to True (the default), then the toolbar buttons will behave like check boxes. If the MultiSelect property is set to False, then the toolbar buttons will behave like radio buttons.

TToolBarButton.AutoWidth Property

```
property AutoWidth: Boolean
```

Available In: Visual Client Applications

Specifies whether the width of the button should be automatically set based upon the Caption, Icon, and Font properties.

TToolBarButton.Caption Property

```
property Caption: String
```

Available In: Visual Client Applications

Specifies the caption to display on the button.

TToolBarButton.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TToolBarButton.Enabled Property

```
property Enabled: Boolean
```

Available In: Visual Client Applications

Specifies whether the button is enabled or disabled. When a button is disabled, it cannot be clicked and is displayed in a disabled state. The default value is True.

TToolBarButton.Font Property

```
property Font: TFont
```

Available In: Visual Client Applications

Specifies the properties of the font used to display the content of the control.

TToolBarButton.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TToolBarButton.Icon Property

```
property Icon: TIconProperties
```

Available In: Visual Client Applications

Specifies the properties of the icon used with the control.

TToolBarButton.Index Property

```
property Index: Integer
```

Available In: Visual Client Applications

Specifies the index of the button in its parent toolbar's buttons.

TToolBarButton.ParentToolBar Property

```
property ParentToolBar: TToolBarControl
```

Available In: Visual Client Applications

Indicates the parent toolbar that contains the button.

TToolBarButton.RepeatClick Property

```
property RepeatClick: Boolean
```

Available In: Visual Client Applications

Specifies whether the OnClick event handler should be triggered every RepeatClickInterval milliseconds while the button is pressed.

TToolBarButton.RepeatClickInterval Property

```
property RepeatClickInterval: Integer
```

Available In: Visual Client Applications

Specifies the interval, in milliseconds, to trigger the OnClick event handler when the RepeatClick is True and the button is pressed.

TToolBarButton.Selected Property

```
property Selected: Boolean
```

Available In: Visual Client Applications

Specifies whether the toolbar button is selected. A toolbar button can only be selected if the AllowSelection property is True. If a toolbar button is selectable, then the Selected property can be modified to toggle the button to and from a "pushed" state.

Note
By default, multiple toolbar buttons within the same toolbar can have their Selected property set to True at the same time. This behavior is controlled by the TToolBar MultiSelect property. If the MultiSelect property is set to True (the default), then the toolbar buttons will behave like check boxes. If the MultiSelect property is set to False, then the toolbar buttons will behave like radio buttons.

TToolBarButton.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TToolBarButton.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TToolBarButton.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

10.236 TToolBarControl Component

Unit: WebTlbrs

Inherits From TControl

Available In: Visual Client Applications

The TToolBarControl control is the base class for toolbar controls, and contains all of the toolbar functionality in the form of public methods and protected properties/events that descendant classes can use to create customized toolbar controls.

Properties	Methods	Events
ButtonCount	MakeButtonVisible	
Buttons	NewButton	
VisibleButtonCount	ScrollNext	
VisibleButtons	ScrollPrior	

TToolBarControl.ButtonCount Property

```
property ButtonCount: Integer
```

Available In: Visual Client Applications

Indicates the number of buttons in the toolbar control.

TToolBarControl.Buttons Property

```
property Buttons[AIndex: Integer]: TToolBarButton
```

Available In: Visual Client Applications

Accesses the buttons in the toolbar control by index.

TToolBarControl.VisibleButtonCount Property

```
property VisibleButtonCount: Integer
```

Available In: Visual Client Applications

Indicates the number of visible buttons in the toolbar control.

TToolBarControl.VisibleButtons Property

```
property VisibleButtons[AIndex: Integer]: TToolBarButton
```

Available In: Visual Client Applications

Accesses the visible buttons in the toolbar control by index.

TToolBarControl.MakeButtonVisible Method

```
procedure MakeButtonVisible(AButton: TToolBarButton)
```

Available In: Visual Client Applications

Use this method to ensure that the specified button is visible.

TToolBarControl.NewButton Method

```
function NewButton: TToolBarButton
```

Available In: Visual Client Applications

Use this method to create a new button. The new button will be positioned after all other existing buttons.

TToolBarControl.ScrollNext Method

```
procedure ScrollNext
```

Available In: Visual Client Applications

Use this method to scroll the buttons to the left so that the left-most button in the control is no longer visible.

TToolBarControl.ScrollPrior Method

```
procedure ScrollPrior
```

Available In: Visual Client Applications

Use this method to scroll the buttons to the right so that the button to the left of the left-most button in the control is visible.

10.237 TVideo Component

Unit: WebMedia

Inherits From TMediaControl

Available In: Visual Client Applications

The TVideo control encapsulates the HTML5 video support available in web browsers. With the TVideo control, you can handle most aspects of video loading and playback.

Properties	Methods	Events
AutoPlay		OnAbort
CurrentTime		OnAnimationComplete
Cursor		OnAnimationsComplete
DataColumn		OnCanPlay
DataSet		OnCanPlayThrough
DefaultPlaybackRate		OnCaptureEnd
Duration		OnCaptureStart
Ended		OnCapturing
Hint		OnClick
Loop		OnContextMenu
Muted		OnDblClick
NetworkState		OnDurationChange
Opacity		OnEmptied
Paused		OnEnded
PlaybackRate		OnError
PosterImageURL		OnHide
Preload		OnLoadedData
ReadyState		OnLoadedMetadata
Seeking		OnLoadStart
ShowControls		OnMouseDown
SourceURL		OnMouseEnter
VideoHeight		OnMouseLeave
VideoWidth		OnMouseMove
Volume		OnMouseUp
		OnMove
		OnPause
		OnPlay
		OnPlaying

		OnProgress
		OnRateChange
		OnSeeked
		OnSeeking
		OnShow
		OnSize
		OnStalled
		OnSuspend
		OnTimeUpdate
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart
		OnVolumeChange
		OnWaiting

TVideo.AutoPlay Property

```
property AutoPlay: Boolean
```

Available In: Visual Client Applications

Specifies that the video should begin playing as soon as enough data has been loaded to allow playback. The default value is False.

TVideo.CurrentTime Property

```
property CurrentTime: Double
```

Available In: Visual Client Applications

Indicates the current playback time, in seconds. Setting this property to a new value will cause the video to skip to the specified time.

TVideo.Cursor Property

```
property Cursor: TCursor
```

Available In: Visual Client Applications

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TVideo.DataColumn Property

```
property DataColumn: String
```

Available In: Visual Client Applications

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TVideo.DataSet Property

```
property DataSet: TDataSet
```

Available In: Visual Client Applications

Specifies the dataset to bind the control to. The default value is nil.

TVideo.DefaultPlaybackRate Property

```
property DefaultPlaybackRate: Double
```

Available In: Visual Client Applications

Specifies the default playback rate, with 1 being normal playback, less than 1 being slower playback, and greater than 1 being faster playback. The default value is 1.

Note
The volume will normally be automatically muted when playing video faster or slower than the normal playback rate.

TVideo.Duration Property

```
property Duration: Double
```

Available In: Visual Client Applications

Indicates the length of the video in seconds. Add an event handler for the OnDurationChange event to detect when the duration has been determined for the current video being loaded/played. If the duration has not been determined, this property will return 0.

TVideo.Ended Property

```
property Ended: Boolean
```

Available In: Visual Client Applications

Indicates that the end of the video has been reached.

TVideo.Hint Property

```
property Hint: String
```

Available In: Visual Client Applications

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TVideo.Loop Property

```
property Loop: Boolean
```

Available In: Visual Client Applications

Specifies that the video playback should automatically restart at the beginning once the end has been reached. The default value is False.

TVideo.Muted Property

```
property Muted: Boolean
```

Available In: Visual Client Applications

Specifies that the playback volume should be muted. The default value is False.

TVideo.NetworkState Property

```
property NetworkState: TMediaNetworkState
```

Available In: Visual Client Applications

Indicates the network state of the video loading/playback.

TVideo.Opacity Property

```
property Opacity: Integer
```

Available In: Visual Client Applications

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TVideo.Paused Property

```
property Paused: Boolean
```

Available In: Visual Client Applications

Indicates that video playback is paused, either by the user pausing the video via the user interface when the ShowControls property is True, or by the application calling the Pause method. The default value is False.

TVideo.PlaybackRate Property

```
property PlaybackRate: Double
```

Available In: Visual Client Applications

Specifies the playback rate, with 1 being normal playback, less than 1 being slower playback, and greater than 1 being faster playback. The default value is 1.

Note
The volume will normally be automatically muted when playing video faster or slower than the normal playback rate.

TVideo.PosterImageURL Property

```
property PosterImageURL: String
```

Available In: Visual Client Applications

Specifies the URL of an image to use as a background before video playback is started.

TVideo.Preload Property

```
property Preload: TMediaPreload
```

Available In: Visual Client Applications

Specifies how much of the current video data should be loaded before playback begins.

TVideo.ReadyState Property

```
property ReadyState: TMediaReadyState
```

Available In: Visual Client Applications

Indicates whether the video is ready for playback, and if so, a general description of what video data has been loaded.

TVideo.Seeking Property

```
property Seeking: Boolean
```

Available In: Visual Client Applications

Indicates that video is switching to a new playback location, either by the user changing the playback location in the video via the user interface when the ShowControls property is True, or by the application setting the CurrentTime property.

TVideo.ShowControls Property

```
property ShowControls: Boolean
```

Available In: Visual Client Applications

Specifies whether the control should show the native user interface for the video being played.

TVideo.SourceURL Property

```
property SourceURL: String
```

Available In: Visual Client Applications

Specifies the URL of the video to be loaded into the control. Whenever this property is changed, the existing video is cleared and the new video will start downloading from the web server. Please review the events available for this control in order to get more information on detecting and handling the loading/playback of the video.

TVideo.VideoHeight Property

```
property VideoHeight: Integer
```

Available In: Visual Client Applications

Indicates the actual height of the video being played.

> **Note**
> This property will be zero until the metadata for the video has been loaded.

TVideo.VideoWidth Property

```
property VideoWidth: Integer
```

Available In: Visual Client Applications

Indicates the actual width of the video being played.

> **Note**
> This property will be zero until the metadata for the video has been loaded.

TVideo.Volume Property

```
property Volume: Integer
```

Available In: Visual Client Applications

Specifies the playback volume of the audio for the video. The volume can be set between 0 and 100.

TVideo.OnAbort Event

```
property OnAbort: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media control has stopped loading data for the current media. This is normally caused by the user requesting such an action via the user interface when the ShowControls property is True.

TVideo.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

Available In: Visual Client Applications

This event is triggered when an animation completes for the control.

TVideo.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when all active animations complete for the control.

TVideo.OnCanPlay Event

```
property OnCanPlay: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media control has loaded enough data to begin playback. However, additional data loading may be required as playback continues.

TVideo.OnCanPlayThrough Event

```
property OnCanPlayThrough: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media control has loaded enough data to begin playback and (probably) play the media until the end without needing to load any additional data.

TVideo.OnCaptureEnd Event

```
property OnCaptureEnd: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered when mouse/touch event capturing is in effect and a mouse up or touch end event occurs for the control.

TVideo.OnCaptureStart Event

```
property OnCaptureStart: TCaptureStartEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse down or touch start event occurs and gives the application an opportunity to begin capturing all subsequent mouse or touch events and routing them to the control. Return True from the event handler to indicate that mouse/touch event capturing should be started for the control.

TVideo.OnCapturing Event

```
property OnCapturing: TCaptureEvent
```

Available In: Visual Client Applications

This event is triggered after mouse/touch event capturing has been started for a control and a mouse move or touch move event occurs.

TVideo.OnClick Event

```
property OnClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TVideo.OnContextMenu Event

```
property OnContextMenu: TContextMenuEvent
```

Available In: Visual Client Applications

This event is triggered when the browser needs to display a context menu for the control. With devices using a mouse, this will occur when the right mouse button is pressed. With devices using a touch interface, this will will occur when a long press occurs on the touch surface. Return True from the event handler to indicate that default browser context menu should be displayed, or False if you wish to use your own custom context menu that you will manually display.

TVideo.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TVideo.OnDurationChange Event

```
property OnDurationChange: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the duration of the media changes, which normally occurs when loading new media into the control by modifying the SourceURL property.

TVideo.OnEmptied Event

```
property OnEmptied: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever an error or abort has caused the NetworkState property to revert to the mnsEmpty state.

TVideo.OnEnded Event

```
property OnEnded: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever playback has stopped because the end of the media has been reached.

TVideo.OnError Event

```
property OnError: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever an error has prevented the media from being loaded properly.

TVideo.OnHide Event

```
property OnHide: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is hidden using the Hide method.

TVideo.OnLoadedData Event

```
property OnLoadedData: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media control has loaded enough data for the current playback location.

TVideo.OnLoadedMetadata Event

```
property OnLoadedMetadata: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media control has loaded the metadata, including the duration and dimensions, for the current media.

TVideo.OnLoadStart Event

```
property OnLoadStart: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media control starts loading data for the current media.

TVideo.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TVideo.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer enters the bounds of the control.

TVideo.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the mouse pointer leaves the bounds of the control.

TVideo.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

Available In: Visual Client Applications

This event is triggered as the mouse pointer is moved over the control.

TVideo.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

Available In: Visual Client Applications

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TVideo.OnMove Event

```
property OnMove: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's position is changed.

TVideo.OnPause Event

```
property OnPause: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media playback is paused.

TVideo.OnPlay Event

```
property OnPlay: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media playback is started/resumed.

TVideo.OnPlaying Event

```
property OnPlaying: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever media playback has actually started.

Note
This event is slightly different from the OnPlay event, which only indicates that the user or application **requested** playback to start/resume. This event may be triggered multiple times during playback, especially if playback needs to stop in order to allow more media data to be loaded, which can be the case with slower network connections.

TVideo.OnProgress Event

```
property OnProgress: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the current media is being loaded.

Note
This event is typically fired several times per second in most web browsers, so be very careful about how time-consuming any event handlers are for this event.

TVideo.OnRateChange Event

```
property OnRateChange: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the playback rate of the media control has changed for the current media. This is caused by the user requesting such an action via the user interface when the ShowControls property is True, or when the application modifies the PlaybackRate property.

TVideo.OnSeeked Event

```
property OnSeeked: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the Seeking property reverts to False.

TVideo.OnSeeking Event

```
property OnSeeking: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the playback location of the media control has changed for the current media. This is caused by the user requesting such an action via the user interface when the ShowControls property is True, or when the application modifies the CurrentTime property.

TVideo.OnShow Event

```
property OnShow: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered when the control is shown using the Show method.

TVideo.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the control's width and/or height are changed.

TVideo.OnStalled Event

```
property OnStalled: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media control is trying to load data for the current media, but no data is arriving over the network.

TVideo.OnSuspend Event

```
property OnSuspend: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media control has loaded enough data to enable playback, and has stopped loading more data.

TVideo.OnTimeUpdate Event

```
property OnTimeUpdate: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the CurrentTime property changes.

Note
This event can be fired as many as 60 times per second in some web browsers, so be very careful about how time-consuming any event handlers are for this event.

TVideo.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TVideo.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control stops being touched via a touch interface.

TVideo.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered as a touch is moved over the control.

TVideo.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

Available In: Visual Client Applications

This event is triggered when the control is touched via a touch interface.

TVideo.OnVolumeChange Event

```
property OnVolumeChange: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the audio volume of the media control has changed for the current media. This is caused by the user requesting such an action via the user interface when the ShowControls property is True, or when the application modifies the Volume property.

TVideo.OnWaiting Event

```
property OnWaiting: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the media control cannot start/resume playback because more data needs to be loaded for the current media.

10.238 TVideoElement Component

Unit: WebUI

Inherits From TMediaElement

Available In: Visual Client Applications

The TVideoElement class is the element class for video UI elements, and contains all of the video playback functionality in the form of public methods and properties/events that control classes can use to create video controls.

> **Note**
> This element does not provide support for video playback at design-time, and the applicable playback methods and properties are all stubs.

Properties	Methods	Events
PosterURL		
VideoHeight		
VideoWidth		

TVideoElement.PosterURL Property

```
property PosterURL: String
```

Available In: Visual Client Applications

Specifies the URL of an image to use as a background before video playback is started.

TVideoElement.VideoHeight Property

```
property VideoHeight: Integer
```

Available In: Visual Client Applications

Indicates the actual height of the video being played.

> **Note**
> This property will be zero until the metadata for the video has been loaded.

TVideoElement.VideoWidth Property

```
property VideoWidth: Integer
```

Available In: Visual Client Applications

Indicates the actual width of the video being played.

> **Note**
> This property will be zero until the metadata for the video has been loaded.

10.239 TViewport Component

Unit: WebForms

Inherits From TComponent

Available In: Visual Client Applications

The TViewport component represents the browser viewport at run-time for a visual application and provides properties for retrieving the browser viewport dimensions, as well as specifying whether browser scrollbars should be displayed when the dimensions of the application surface overflows the viewport's client area.

Properties	Methods	Events
Height	ScrollBy	OnScroll
OverflowX		OnSize
OverflowY		
ResizeDelay		
ScrollLeft		
ScrollTop		
Width		

TViewport.Height Property

```
property Height: Integer
```

Available In: Visual Client Applications

Indicates the height of the browser viewport.

TViewport.OverflowX Property

```
property OverflowX: TOverflowType
```

Available In: Visual Client Applications

Specifies whether or not to show a native horizontal browser scrollbar for the application if the width of its surface exceeds the width of the client rectangle for the element.

TViewport.OverflowY Property

```
property OverflowY: TOverflowType
```

Available In: Visual Client Applications

Specifies whether or not to show a native vertical browser scrollbar for the application if the height of its surface exceeds the height of the client rectangle for the element.

TViewport.ResizeDelay Property

```
property ResizeDelay: Integer
```

Available In: Visual Client Applications

Specifies how long, in milliseconds, the application will wait after a browser viewport resize before updating the application's user interface.

TViewport.ScrollLeft Property

```
property ScrollLeft: Integer
```

Available In: Visual Client Applications

If an application's Surface width and/or height is greater than the application viewport's Width and Height properties, then this property indicates the amount, in pixels, that the browser viewport has been scrolled to the right.

Specify a new value to manually scroll the browser viewport to the left or right. A value of 0 means that the viewport is scrolled all the way to its left-most position.

TViewport.ScrollTop Property

```
property ScrollTop: Integer
```

Available In: Visual Client Applications

If an application's Surface width and/or height is greater than the application viewport's Width and Height properties, then this property indicates the amount, in pixels, that the browser viewport has been scrolled towards the bottom.

Specify a new value to manually scroll the browser viewport towards the top or bottom. A value of 0 means that the viewport is scrolled all the way to its top-most position.

TViewport.Width Property

```
property Width: Integer
```

Available In: Visual Client Applications

Indicates the width of the browser viewport.

TViewport.ScrollBy Method

```
procedure ScrollBy(X,Y: Integer)
```

Available In: Visual Client Applications

If an application's Surface width and/or height is greater than the application viewport's Width and Height properties, then you can use this method to scroll the viewport of the application horizontally, vertically, or both. The X and Y values represent the number of pixels to scroll the viewport by, and may be negative values for scrolling backward.

TViewport.OnScroll Event

```
property OnScroll: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the browser viewport is scrolled horizontally or vertically.

TViewport.OnSize Event

```
property OnSize: TNotifyEvent
```

Available In: Visual Client Applications

This event is triggered whenever the browser viewport's width and/or height are changed.

10.240 TWebControl Component

Unit: WebBrwsr

Inherits From TBindableColumnControl

Available In: Visual Client Applications

The TWebControl control is the base class for web browser controls that dynamically load resources, and contains all of the web browser control functionality in the form of public methods and protected properties/events that descendant classes can use to create customized web browser controls.

Properties	Methods	Events
	Refresh	

TWebControl.Refresh Method

```
procedure Refresh
```

Available In: Visual Client Applications

Use this method to refresh a resource without having to modify the URL. This is useful when the resource changes on the server, but the URL does not.

10.241 TWebElement Component

Unit: WebUI

Inherits From TElement

Available In: Visual Client Applications

The TWebElement class is the base element class for browser objects that can be dynamically-loaded as a resource, and contains all of the browser object functionality in the form of public methods and properties/events that control classes can use to create browser object controls.

> **Note**
> This element does not provide support for browser objects at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events
Loaded	Refresh	OnError
URL		OnLoad
		OnUnload

TWebElement.Loaded Property

```
property Loaded: Boolean
```

Available In: Visual Client Applications

Indicates whether the object resource specified by the URL property has been loaded.

An event handler can be attached to the OnLoad event to execute code when the object is loaded.

TWebElement.URL Property

```
property URL: String
```

Available In: Visual Client Applications

Specifies the URL for the object resource. Whenever the URL property changes, the OnUnload event is triggered immediately. The OnLoad event is triggered once the object resource has been loaded.

TWebElement.Refresh Method

```
procedure Refresh
```

Available In: Visual Client Applications

Use this method to refresh a resource without having to modify the URL property. This is useful when the resource changes on the server, but the URL does not.

TWebElement.OnError Event

```
property OnError: TWebElementEvent
```

Available In: Visual Client Applications

This event is triggered an error occurs while loading the resource specified by the URL property.

TWebElement.OnLoad Event

```
property OnLoad: TWebElementEvent
```

Available In: Visual Client Applications

This event is triggered when the resource specified by the URL property has been completely loaded.

TWebElement.OnUnload Event

```
property OnUnload: TWebElementEvent
```

Available In: Visual Client Applications

This event is triggered when the currently-loaded resource specified by the URL property has been unloaded.

10.242 TWebServerRequest Component

Unit: WebSrvr

Inherits From THTTPServerRequest

Available In: Server Applications

The TWebServerRequest class is a descendant of the THTTPServerRequest class and represents an incoming web server request in server applications. In addition to the base properties and methods, this class adds access to the web server session that was established during user authentication (before the current web server request).

Properties	Methods	Events
RequestSession	SetResponseNoCacheHeader	

TWebServerRequest.RequestSession Property

```
property RequestSession: TWebServerSession
```

Available In: Server Applications

Provides access to the session associated with the web server request.

TWebServerRequest.SetResponseNoCacheHeader Method

```
procedure SetResponseNoCacheHeader
```

Available In: Server Applications

Use this method to set the **Cache-Control** response header to "no-cache" in the ResponseHeaders property.

10.243 TWebServerSession Component

Unit: WebSrvr

Inherits From TExternalObject

Available In: Server Applications

The TWebServerSession class represents the web server session that was established during user authentication and contains information about the authenticated user and the web server session itself.

> **Note**
> A web server session must be established and the user authenticated before any request can be made to a server application. This occurs transparently when the client application makes the proper authentication calls to the web server, and failure to do so will result in any calls to a server application failing.

Properties	Methods	Events
Authenticated	CommitSchemaTransaction	
Expiration	HasPrivilege	
ID	HasRole	
UserFullName	LoadRowSet	
UserName	LoadRowSetColumns	
Variables		

TWebServerSession.Authenticated Property

```
property Authenticated: Boolean
```

Available In: Server Applications

Indicates whether the session has been authenticated.

TWebServerSession.Expiration Property

```
property Expiration: DateTime
```

Available In: Server Applications

Indicates when the session will expire.

TWebServerSession.ID Property

```
property ID: String
```

Available In: Server Applications

Indicates the session ID.

TWebServerSession.UserFullName Property

```
property UserFullName: String
```

Available In: Server Applications

Indicates the full name of the authenticated user associated with the session.

TWebServerSession.UserName Property

```
property UserName: String
```

Available In: Server Applications

Indicates the user name of the authenticated user associated with the session.

TWebServerSession.Variables Property

```
property Variables: TWebServerSessionVariables
```

Available In: Server Applications

Provides access to session variables, which is a dictionary of keys and values that server applications can use to persist data for a session.

TWebServerSession.CommitSchemaTransaction Method

```
procedure CommitSchemaTransaction(const ADatabaseName: String;
      ASchema: TSchema)
```

Available In: Server Applications

This method is for internal use with database transactions within server applications. Do not call this method.

TWebServerSession.HasPrivilege Method

```
function HasPrivilege(const AName: String): Boolean
```

Available In: Server Applications

Use this method to determine if the authenticated user associated with the session has been granted one or more roles that contains the specified privilege.

TWebServerSession.HasRole Method

```
function HasRole(const AName: String): Boolean
```

Available In: Server Applications

Use this method to determine if the authenticated user associated with the session has been granted the specified role.

TWebServerSession.LoadRowSet Method

```
procedure LoadRowSet(const ADatabaseName: String; ARowSet:
      TRowSet; AParameters: THTTPParameters=nil)
```

Available In: Server Applications

This method is for internal use with datasets within server applications. Do not call this method.

TWebServerSession.LoadRowSetColumns Method

```
procedure LoadRowSetColumns(const ADatabaseName: String;
      ARowSet: TRowSet; AParameters: THTTPParameters=nil)
```

Available In: Server Applications

This method is for internal use with datasets within server applications. Do not call this method.

10.244 TWebServerSessionVariables Component

Unit: WebSrvr

Inherits From TExternalObject

Available In: Server Applications

The TWebServerSessionVariables class represents the user-defined session variables for a given web server session. The session variables are variant values associated with specific string keys and have a lifetime equivalent to the web server session. This means that you can store data here that will be accessible to server applications across multiple web server requests.

> **Warning**
> Do not store class instance, class, method, or function/procedure references as session variables, as all of these types of data will not outlive the session and will most likely represent an invalid reference when accessed in subsequent web server requests.

Properties	Methods	Events
Count	Clear	
Value	Create	
	DeleteValue	
	GetKeys	
	GetValue	
	SetValue	
	ValueExists	

TWebServerSessionVariables.Count Property

```
property Count: Integer
```

Available In: Server Applications

Indicates the total number of variables contained within the list of variables.

TWebServerSessionVariables.Value Property

```
property Value[const AKey: String]: Variant
```

Available In: Server Applications

Gets or sets the value of the variable that matches the specified name.

When getting the value: if a defined variable does not exist with the specified name, then the method will return a Null (variant) value.

When setting the value: if a variable with the specified name is already defined, then its value will be updated. If a variable with the the specified name does not already exist, then it will be added.

TWebServerSessionVariables.Clear Method

```
procedure Clear
```

Available In: Server Applications

Use this method to delete all defined variables.

TWebServerSessionVariables.Create Method

```
constructor Create(ASize: Integer)
```

Available In: Server Applications

Use this method to create a new instance of the TWebServerSessionVariables class.

> **Note**
> Because the variables are automatically created and included as part of the session associated with the incoming web server request, you will most likely never need to call this method.

TWebServerSessionVariables.DeleteValue Method

```
procedure DeleteValue(const AKey: String)
```

Available In: Server Applications

Use this method to delete the variable with the specified name from the defined variables.

TWebServerSessionVariables.GetKeys Method

```
function GetKeys: array of String
```

Available In: Server Applications

Use this method to get an array of strings that contains the names of all of the defined variables.

TWebServerSessionVariables.GetValue Method

```
function GetValue(const AKey: String): Variant
```

Available In: Server Applications

Use this method to get the value for the variable with the specified name. If a defined variable does not exist with the specified name, then the method will return a Null (variant) value.

TWebServerSessionVariables.SetValue Method

```
procedure SetValue(const AKey: String; const AValue: Variant)
```

Available In: Server Applications

Use this method to set the value for the variable with the specified name. If the specified variable is already defined, then its value will be updated. If the specified variable does not already exist, then it will be added.

TWebServerSessionVariables.ValueExists Method

```
function ValueExists(const AKey: String): Boolean
```

Available In: Server Applications

Use this method to determine if a variable with the specified name exists in the defined variables.

10.245 TWriter Component

Unit: WebCore

Inherits From TObject

Available In: Client and Server Applications

The TWriter class is a class used by TPersistent-descendant classes to save their published properties to JSON strings. It can be used as a general-purpose JSON writer in your applications.

When a TWriter instance is created, the constructor allows you to specify:

- The date-time format to use when writing date-time properties.
- The number of spaces to use per indentation level, if the output is not compressed.
- Whether to include line feeds in the JSON output, if the output is not compressed.
- Whether to compress all whitespace in the JSON output. Compressing the whitespace removes any unnecessary whitespace in order to keep the size of the JSON output to a minimum.

Properties	Methods	Events
Output	ArrayProperty	
	BeginArray	
	BeginNewLine	
	BeginObject	
	BooleanProperty	
	BooleanValue	
	CancelNewLine	
	Create	
	DateTimeProperty	
	DateTimeValue	
	DecIndent	
	EndArray	
	EndObject	
	FloatProperty	
	FloatValue	
	IncIndent	
	Initialize	
	IntegerProperty	
	IntegerValue	
	Literal	

	NewLine	
	NullProperty	
	NullValue	
	ObjectProperty	
	PropertyName	
	Separator	
	StringProperty	
	StringValue	
	Whitespace	

TWriter.Output Property

```
property Output: String
```

Available In: Client and Server Applications

Indicates the current JSON output.

TWriter.ArrayProperty Method

```
procedure ArrayProperty(const Name: String)
```

Available In: Client and Server Applications

Use this method to write an array property's name using the PropertyName method.

TWriter.BeginArray Method

```
procedure BeginArray
```

Available In: Client and Server Applications

Use this method to begin writing a new array.

TWriter.BeginNewLine Method

```
procedure BeginNewLine
```

Available In: Client and Server Applications

Use this method to set a flag requesting that the next property should be written on a new line.

> **Note**
> New lines are not used if the JSON output is being compressed.

TWriter.BeginObject Method

```
procedure BeginObject
```

Available In: Client and Server Applications

Use this method to begin writing a new object.

TWriter.BooleanProperty Method

```
procedure BooleanProperty(const Name: String; Value: Boolean)
```

Available In: Client and Server Applications

Use this method to write a boolean property.

TWriter.BooleanValue Method

```
procedure BooleanValue(Value: Boolean)
```

Available In: Client and Server Applications

Use this method to write a boolean value.

TWriter.CancelNewLine Method

```
procedure CancelNewLine
```

Available In: Client and Server Applications

Use this method to cancel a new line started via the NewLine method. This is useful if a new line is started for a property, but the property cannot be written for some reason.

TWriter.Create Method

```
constructor Create(ADateTimeFormat: TDateTimeFormat=dtfRaw;
      AIndentSpaces: Integer=3; AIncludeLineFeeds: Boolean=True;
      ACompressWhitespace: Boolean=False)
```

Available In: Client and Server Applications

Use this method to create a new instance of the TWriter class. The optional ADateTimeFormat parameter indicates whether date and time values should be output as an ISO 8601 date and time string value, or as a raw Unix date and time integer value (the number of milliseconds since midnight, January 1, 1970). The optional AIndentSpaces parameter indicates how many spaces to use for indentation in the output, the optional AIncludeLineFeeds parameter indicates whether to include line feeds (CRLF) in the output, and the optional ACompressWhitespace parameter indicates whether any whitespace should be compressed (removed) from the output.

TWriter.DateTimeProperty Method

```
procedure DateTimeProperty(const Name: String; Value: DateTime)
```

Available In: Client and Server Applications

Use this method to write a date-time property. How a date-time property is written is controlled by the first TDateTimeFormat parameter in the TWriter class constructor.

TWriter.DateTimeValue Method

```
procedure DateTimeValue(Value: DateTime)
```

Available In: Client and Server Applications

Use this method to write a date-time value. How a date-time value is written is controlled by the first TDateTimeFormat parameter in the TWriter class constructor.

TWriter.DecIndent Method

```
procedure DecIndent
```

Available In: Client and Server Applications

Decrement the indentation level for the output.

> **Note**
> Indentation levels are not used if the JSON output is being compressed.

TWriter.EndArray Method

```
procedure EndArray
```

Available In: Client and Server Applications

Use this method to end writing an array.

TWriter.EndObject Method

```
procedure EndObject
```

Available In: Client and Server Applications

Use this method to end writing an object.

TWriter.FloatProperty Method

```
procedure FloatProperty(const Name: String; Value: Double)
```

Available In: Client and Server Applications

Use this method to write a float property.

TWriter.FloatValue Method

```
procedure FloatValue(Value: Double)
```

Available In: Client and Server Applications

Use this method to write a float value.

TWriter.IncIndent Method

```
procedure IncIndent
```

Available In: Client and Server Applications

Increment the indentation level for the JSON output.

> **Note**
> Indentation levels are not used if the JSON output is being compressed.

TWriter.Initialize Method

```
procedure Initialize
```

Available In: Client and Server Applications

Use this method to initialize the writer so that the Output property is blank.

TWriter.IntegerProperty Method

```
procedure IntegerProperty(const Name: String; Value: Integer)
```

Available In: Client and Server Applications

Use this method to write an integer property.

TWriter.IntegerValue Method

```
procedure IntegerValue(Value: Integer)
```

Available In: Client and Server Applications

Use this method to write an integer value.

TWriter.Literal Method

```
procedure Literal(const Value: String)
```

Available In: Client and Server Applications

Use this method to write a literal.

TWriter.NewLine Method

```
procedure NewLine
```

Available In: Client and Server Applications

Use this method to write a new line (CRLF).

> **Note**
> New lines are not written if the JSON output is being compressed, or if the writer was created with the new line option turned off.

TWriter.NullProperty Method

```
procedure NullProperty(const Name: String)
```

Available In: Client and Server Applications

Use this method to write a null property.

TWriter.NullValue Method

```
procedure NullValue
```

Available In: Client and Server Applications

Use this method to write a null value.

TWriter.ObjectProperty Method

```
procedure ObjectProperty(const Name: String)
```

Available In: Client and Server Applications

Use this method to write an object property's name using the PropertyName method.

TWriter.PropertyName Method

```
procedure PropertyName(const Name: String)
```

Available In: Client and Server Applications

Use this method to write a property name.

TWriter.Separator Method

```
procedure Separator
```

Available In: Client and Server Applications

Use this method to write a separator (,).

TWriter.StringProperty Method

```
procedure StringProperty(const Name: String; const Value:
      String)
```

Available In: Client and Server Applications

Use this method to write a string property.

TWriter.StringValue Method

```
procedure StringValue(const Value: String)
```

Available In: Client and Server Applications

Use this method to write a string value.

TWriter.Whitespace Method

```
procedure Whitespace
```

Available In: Client and Server Applications

Use this method to write a space ( ).

> **Note**
> Whitespace is not written if the JSON output is being compressed.

10.246 TZoomControlOptions Component

Unit: WebMaps

Inherits From TMapOption

Available In: Visual Client Applications

The TZoomControlOptions class controls how the zoom control is configured in a TMap control. These zoom control options correspond to the zoom control options available for maps in the Google Maps API.

Properties	Methods	Events
Position		
Style		

TZoomControlOptions.Position Property

```
property Position: TMapControlPosition
```

Available In: Visual Client Applications

Specifies the position of the zoom control.

TZoomControlOptions.Style Property

```
property Style: TZoomControlStyle
```

Available In: Visual Client Applications

Specifies the style of the zoom control.

This page intentionally left blank

Chapter 11
Type Reference

11.1 TAlertOrientation Type

Unit: WebLabels

```
TAlertOrientation = (aoLeft,aoRight)
```

Available In: Client Applications

The TAlertOrientation enumerated type is used with the TAlertLabel component to specify the orientation of the label's caption.

Element	Description
aoLeft	The label's caption will be positioned at the left side of the label.
aoRight	The label's caption will be positioned at the right side of the label.

11.2 TAnimatedIconDirection Type

Unit: WebIcons

```
TAnimatedIconDirection = (idVertical,idHorizontal)
```

Available In: Client Applications

The TAnimatedIconDirection enumerated type is used with the TAnimatedIcon component to specify the direction in which the frames of the animated icon are laid out.

Element	Description
idHorizontal	Specifies that the frames are laid out in a horizontal direction.
idVertical	Specifies that the frames are laid out in a vertical direction.

11.3 TAnimationCompleteEvent Type

Unit: WebCtrls

```
TAnimationCompleteEvent = procedure (Sender: TObject; Animation:
      TAnimation) of object
```

Available In: Client Applications

The TAnimationCompleteEvent type is a common event type that is used by controls to provide notification that an animation has completed for the control.

The Sender parameter represents the class instance that triggered the event. The Animation parameter represents the TAnimation instance that has completed.

11.4 TAnimationStyle Type

Unit: WebUI

```
TAnimationStyle = (asNone,asLinear,asQuadEaseOut,asQuadEaseIn,
      asQuadEaseInOut,asQuadEaseOutIn,asExpoEaseOut, asExpoEaseIn,
      asExpoEaseInOut,asExpoEaseOutIn, asCubicEaseOut,asCubicEaseIn,
      asCubicEaseInOut, asCubicEaseOutIn,asQuartEaseOut,asQuartEaseIn,
      asQuartEaseInOut,asQuartEaseOutIn,asQuintEaseOut, asQuintEaseIn,
      asQuintEaseInOut,asQuintEaseOutIn, asCircEaseOut,asCircEaseIn,
      asCircEaseInOut,asCircEaseOutIn, asSineEaseOut,asSineEaseIn,
      asSineEaseInOut,asSineEaseOutIn, asElasticEaseOut,
      asElasticEaseIn,asElasticEaseInOut, asElasticEaseOutIn,
      asBounceEaseOut,asBounceEaseIn, asBounceEaseInOut,
      asBounceEaseOutIn,asBackEaseOut, asBackEaseIn,asBackEaseInOut,
      asBackEaseOutIn)
```

Available In: Client Applications

The TAnimationStyle enumerated type is used to specify how an animation should transform a given property of a UI element or control.

Element	Description
asBackEaseIn	Easing equation function for a back easing in: accelerating from zero velocity.
asBackEaseInOut	Easing equation function for a back easing in/out: acceleration until halfway, then deceleration.
asBackEaseOut	Easing equation function for a back easing out: decelerating from zero velocity.
asBackEaseOutIn	Easing equation function for a back easing out/in: deceleration until halfway, then acceleration.
asBounceEaseIn	Easing equation function for a bounce (exponentially decaying parabolic bounce) easing in: accelerating from zero velocity.
asBounceEaseInOut	Easing equation function for a bounce (exponentially decaying parabolic bounce) easing in/out: acceleration until halfway, then deceleration.
asBounceEaseOut	Easing equation function for a bounce (exponentially decaying parabolic bounce) easing out: decelerating from zero velocity.
asBounceEaseOutIn	Easing equation function for a bounce (exponentially decaying parabolic bounce) easing out/in: deceleration until halfway, then acceleration.
asCircEaseIn	Easing equation function for a circular easing in: accelerating from zero velocity.
asCircEaseInOut	Easing equation function for a circular easing in/out: acceleration until halfway, then deceleration.
asCircEaseOut	Easing equation function for an exponential easing out/in: deceleration until halfway, then acceleration.

asCircEaseOutIn	Easing equation function for a circular easing in/out: acceleration until halfway, then deceleration.
asCubicEaseIn	Easing equation function for a cubic easing in: accelerating from zero velocity.
asCubicEaseInOut	Easing equation function for a cubic easing in/out: acceleration until halfway, then deceleration.
asCubicEaseOut	Easing equation function for a cubic easing out: decelerating from zero velocity.
asCubicEaseOutIn	Easing equation function for a cubic easing out/in: deceleration until halfway, then acceleration.
asElasticEaseIn	Easing equation function for an elastic (exponentially decaying sine wave) easing in: accelerating from zero velocity.
asElasticEaseInOut	Easing equation function for an elastic (exponentially decaying sine wave) easing in/out: acceleration until halfway, then deceleration.
asElasticEaseOut	Easing equation function for an elastic (exponentially decaying sine wave) easing out: decelerating from zero velocity.
asElasticEaseOutIn	Easing equation function for an elastic (exponentially decaying sine wave) easing out/in: deceleration until halfway, then acceleration.
asExpoEaseIn	Easing equation function for an exponential easing in: accelerating from zero velocity.
asExpoEaseInOut	Easing equation function for an exponential easing in/out: acceleration until halfway, then deceleration.
asExpoEaseOut	Easing equation function for an exponential easing out: decelerating from zero velocity.
asExpoEaseOutIn	Easing equation function for an exponential easing out/in: deceleration until halfway, then acceleration.
asLinear	Easing equation function for a simple linear tweening, with no easing.
asNone	No animation style.
asQuadEaseIn	Easing equation function for a quadratic easing in: accelerating from zero velocity.
asQuadEaseInOut	Easing equation function for a quadratic easing in/out: acceleration until halfway, then deceleration.
asQuadEaseOut	Easing equation function for a quadratic easing out: decelerating from zero velocity.
asQuadEaseOutIn	Easing equation function for a quadratic easing out/in: deceleration until halfway, then acceleration.
asQuartEaseIn	Easing equation function for a quartic easing in: accelerating from zero velocity.
asQuartEaseInOut	Easing equation function for a quartic easing in/out: acceleration until halfway, then deceleration.
asQuartEaseOut	Easing equation function for a quartic easing out: decelerating from zero velocity.
asQuartEaseOutIn	Easing equation function for a quartic easing out/in: deceleration until halfway, then acceleration.

asQuintEaseIn	Easing equation function for a quintic easing in: accelerating from zero velocity.
asQuintEaseInOut	Easing equation function for a quintic easing in/out: acceleration until halfway, then deceleration.
asQuintEaseOut	Easing equation function for a quintic easing out: decelerating from zero velocity.
asQuintEaseOutIn	Easing equation function for a quintic easing in/out: acceleration until halfway, then deceleration.
asSineEaseIn	Easing equation function for a sinusoidal easing in: accelerating from zero velocity.
asSineEaseInOut	Easing equation function for a sinusoidal easing in/out: acceleration until halfway, then deceleration.
asSineEaseOut	Easing equation function for a sinusoidal easing out: decelerating from zero velocity.
asSineEaseOutIn	Easing equation function for a sinusoidal easing in/out: deceleration until halfway, then acceleration.

11.5 TAutoCompleteType Type

Unit: WebUI

```
TAutoCompleteType = (acDefault,acOn,acOff)
```

Available In: Client Applications

The TAutoCompleteType enumerated type is used with the descendant components of the TEditControl class to specify how auto-completion should be handled for the control. Auto-completion allows the browser to display a list of suggestions for input values, based upon earlier input values entered by the user.

Element	Description
acDefault	Specifies that auto-completion is enabled or disabled according to the default browser setting.
acOff	Specifies that auto-completion should be disabled.
acOn	Specifies that auto-completion should be enabled.

11.6 TBackgroundImageAnimateDirection Type

Unit: WebUI

```
TBackgroundImageAnimateDirection = (idVertical,idHorizontal)
```

Available In: Client Applications

The TBackgroundImageAnimateDirection enumerated type is used with the TBackgroundImage BeginAnimation method to specify in which direction the background image should be animated.

> **Note**
> The specified animation direction should match the actual orientation of the animation frames in the background image. If it doesn't match, then the animation will not appear correctly.

Element	Description
idHorizontal	Specifies that the background image should be animated in a horizontal direction.
idVertical	Specifies that the background image should be animated in a vertical direction.

11.7 TBackgroundImagePositionType Type

Unit: WebUI

```
TBackgroundImagePositionType = (ptSpecified,ptTopLeft,
      ptTopCenter,ptTopRight, ptCenterLeft,ptCenterCenter,
      ptCenterRight, ptBottomLeft,ptBottomCenter,ptBottomRight)
```

Available In: Client Applications

The TBackgroundImagePositionType enumerated type is used with the TBackgroundImage class to specify how background images should be positioned for UI elements and controls.

Element	Description
ptBottomCenter	Specifies that the background image should be centered horizontally at the bottom of the containing element or control.
ptBottomLeft	Specifies that the background image should be displayed at the bottom-left of the containing element or control.
ptBottomRight	Specifies that the background image should be displayed at the bottom-right of the containing element or control.
ptCenterCenter	Specifies that the background image should be centered horizontally and vertically within the containing element or control.
ptCenterLeft	Specifies that the background image should be centered vertically at the left of the containing element or control.
ptCenterRight	Specifies that the background image should be centered vertically at the right of the containing element or control.
ptSpecified	Specifies that the background image should be positioned according to the TBackgroundImage Left and Top properties.
ptTopCenter	Specifies that the background image should be centered horizontally at the top of the containing element or control.
ptTopLeft	Specifies that the background image should be displayed at the top-left of the containing element or control.
ptTopRight	Specifies that the background image should be displayed at the top-right of the containing element or control.

11.8 TBackgroundImageRepeatStyle Type

Unit: WebUI

```
TBackgroundImageRepeatStyle = (rsBoth,rsHorizontal,rsVertical,
      rsNone)
```

Available In: Client Applications

The TBackgroundImageRepeatStyle enumerated type is used with the TBackgroundImage class to specify how background images should be tiled, if at all, for UI elements and controls.

> **Note**
> The TBackgroundImage PositionType and SizeType properties, as well as the TBackground Origin and Clip properties, will affect how the background image is tiled.

Element	Description
rsBoth	Specifies that the background image should be tiled in both a horizontal and vertical direction.
rsHorizontal	Specifies that the background image should be tiled in a horizontal direction only.
rsNone	Specifies that the background image should not be tiled.
rsVertical	Specifies that the background image should be tiled in a vertical direction only.

11.9 TBackgroundImageSizeType Type

Unit: WebUI

```
TBackgroundImageSizeType = (stNone,stSpecified,stContain,
      stCover)
```

Available In: Client Applications

The TBackgroundImageSizeType enumerated type is used with the TBackgroundImage class to specify how background images should be sized for UI elements and controls.

Element	Description
stContain	Specifies that the background image should be proportionally sized so that it fits within the bounds of the element or control.
stCover	Specifies that the background image should be proportionally sized so that the image covers the bounds of the element or control.
stNone	Specifies that the background image should not be sized and should remain its original size.
stSpecified	Specifies that the background image should be sized according to the TBackgroundImage Width and Height properties.

11.10 TBackgroundOrientationType Type

Unit: WebUI

```
TBackgroundOrientationType = (otBounds,otBorder,otClient)
```

Available In: Client Applications

The TBackgroundOrientationType enumerated type is used with the TBackground class to specify how backgrounds should be positioned and clipped for UI elements and controls.

Element	Description
otBorder	Specifies that the background should originate and/or be clipped based upon the bounding rectangle of the UI element or control, minus any border.
otBounds	Specifies that the background should originate and/or be clipped based upon the bounding rectangle of the UI element or control.
otClient	Specifies that the background should originate and/or be clipped based upon the client rectangle of the border of the UI element or control. The client rectangle is the bounding rectangle minus any border or padding.

11.11 TBalloonOrientation Type

Unit: WebLabels

```
TBalloonOrientation = (boLeft,boCenter,boRight)
```

Available In: Client Applications

The TBalloonOrientation enumerated type is used with the TBalloonLabel component to specify the orientation of the balloon tail.

Element	Description
boCenter	The balloon tail will be positioned centered at the bottom of the label.
boLeft	The balloon tail will be positioned at the bottom-left corner of the label.
boRight	The balloon tail will be positioned at the bottom-right corner of the label.

11.12 TBooleanArray Type

Unit: WebCore

```
TBooleanArray = array of Boolean
```

Available In: Client and Server Applications

The TBooleanArray type is used in classes such as the TSet class to represent an array of Boolean values.

11.13 TBorderStyle Type

Unit: WebUI

```
TBorderStyle = (bsSolid)
```

Available In: Client Applications

The TBorderStyle enumerated type is used with the TBorder class to specify the border style of one side of a border for UI elements and controls.

> **Note**
> This enumerated type currently has only one value.

Element	Description
bsSolid	Specifies that the border should be a solid line.

11.14 TCalendarView Type

Unit: WebCals

```
TCalendarView = (cvMonth,cvYear,cvDecade,cvCentury)
```

Available In: Client Applications

The TCalendarView enumerated type is used with the descendant components of the TCalendarControl class to specify the active view for the calendar.

Element	Description
cvCentury	The calendar control will show a list of decades in the current century.
cvDecade	The calendar control will show a list of years in the current decade.
cvMonth	The calendar control will show a list of days in the current month (the default).
cvYear	The calendar control will show a list of months in the current year.

11.15 TCanPlayMedia Type

Unit: WebUI

```
TCanPlayMedia = (cpmCannot,cpmMaybe,cpmProbably)
```

Available In: ClientApplications

The TCanPlayMedia enumerated type is used with the TMediaControl CanPlayMedia method and TMediaElement CanPlayMedia method to determine if a particular type of media can be played.

Element	Description
cpmCannot	The media can definitely not be played by the web browser.
cpmMaybe	The media may be able to be played by the web browser, but the browser is not certain.
cpmProbably	The media should be able to be played by the web browser.

11.16 TCanvasPoints Type

Unit: WebUI

```
TCanvasPoints = array of TCanvasPoint
```

Available In: Client Applications

The TCanvasPoints type is used in canvas drawing operations to represent an array of TCanvasPoint instances.

11.17 TCaption Type

Unit: WebLabels

```
TCaption = type String
```

Available In: Client Applications

The TCaption type is used to represent the caption of a control. This type is type-equivalent to a String type, but is used to distinguish the caption when used with special design-time property editors in order to allow for multi-line captions.

11.18 TCaptureEvent Type

Unit: WebCtrls

```
TCaptureEvent = procedure (Sender: TObject; X,Y: Integer) of
      object
```

Available In: Client Applications

The TCaptureEvent type is a common event type used by the OnCapturing and OnCaptureEnd properties of a control when mouse pointer or touch input capturing is in effect for the control.

The Sender parameter represents the class instance that triggered the event. The X and Y parameters indicate the horizontal and vertical position of the mouse pointer or touch input, in pixels, relative to the bounds of the control that triggered the event.

11.19 TCaptureStartEvent Type

Unit: WebCtrls

```
TCaptureStartEvent = function (Sender: TObject; Button: Integer;
      ShiftKey, CtrlKey, AltKey: Boolean; X,Y: Integer): Boolean of
      object
```

Available In: Client Applications

The TCaptureStartEvent type is a common event type used by the OnCaptureStart properties of a control when the mouse pointer is pressed or touch input occurs, and gives the control an opportunity to capture the mouse pointer or touch input.

The Sender parameter represents the class instance that triggered the event. The Button parameter represents the ordinal button code of the mouse button pressed, and the ShiftKey, CtrlKey, AltKey parameters represent whether the Shift, Control, and/or Alt keys were also pressed. The X and Y parameters indicate the horizontal and vertical position of the mouse pointer or touch input, in pixels, relative to the bounds of the control that triggered the event.

The ordinal mouse button values are defined in the WebUI unit, and are as follows:

```
MB_NONE = 0;
MB_LEFT = 1;
MB_MIDDLE = 2;
MB_RIGHT = 3;
```

11.20 TCharArray Type

Unit: WebCore

```
TCharArray = array of Char
```

Available In: Client and Server Applications

The TCharArray type is used in classes such as the TStringBuilder class to represent an array of Char values.

11.21 TClickEvent Type

Unit: WebCtrls

```
TClickEvent = function (Sender: TObject): Boolean of object
```

Available In: Client Applications

The TClickEvent type is a common event type that is used by controls to provide notification that a sub-control has been clicked. For example, the TEditComboBox OnButtonClick event is used to allow the developer to intercept when the combo button is clicked in the control.

The Sender parameter represents the class instance that triggered the event.

To not allow the click, return False as the result to any event handler attached to this event. To allow the click, return True.

11.22 TCloseQueryEvent Type

Unit: WebCtrls

```
TCloseQueryEvent = function (Sender: TObject): Boolean of object
```

Available In: Client Applications

The TCloseQueryEvent type is used by the OnCloseQuery event for the TPanel, TForm, and TDialog components to intercept the closing of the control.

Return True as the result of the event to allow the close to continue, or False to prevent the control from closing.

11.23 TCollectionItemClass Type

Unit: WebCore

```
TCollectionItemClass = class of TCollectionItem
```

Available In: Client and Server Applications

The TCollectionItemClass type is used to represent a TCollectionItem class type, and allows TCollectionItemClass variables to store references to TCollectionItem classes and descendants.

11.24 TCollectionItemName Type

Unit: WebCore

```
TCollectionItemName = type String
```

Available In: Client and Server Applications

The TCollectionItemName type is used to represent the name of a TCollectionItem class instance. This type is type-equivalent to a String type, but is used to distinguish the name of a collection item instance when used with special design-time property editors.

11.25 TColor Type

Unit: WebUI

```
TColor = type Integer
```

Available In: Client and Server Applications

The TColor type is an Integer type used to represent a 32-bit RGBA color. The components of the RGBA value are as follows:

Component	Description
Red	The blue component is stored in the lower 8 bits of the 32-bit RGBA value (0-7).
Green	The green component is stored in the next 8 bits of the 32-bit RGBA value (8-15).
Blue	The red component is stored in the next 8 bits of the 32-bit RGBA value (16-23).
Alpha	The alpha (opacity) component is stored in the last 8 bits of the 32-bit RGBA value (24-31).

11.26 TColorSpace Type

Unit: WebSrvr

```
TColorSpace = (csUnknown,csRGB,csCMYK,csGRAY,csYCBCR,csYCCK)
```

Available In: Client and Server Applications

The TColorSpace enumerated type is used to identity the type of color space used by the image contained within a TRasterImage instance.

Element	Description
csCMYK	The image uses the CMYK (Cyan-Magenta-Yellow-Black) color space.
csGRAY	The image uses the grayscale color space.
csRGB	The image uses the RGB (Red-Green-Blue) color space.
csUnknown	The image uses an unknown type of color space.
csYCBCR	The image uses the YCbCr (Y-Luma, Cb-Blue difference chroma, Cr-Red difference chroma) color space.
csYCCK	The image uses the YCCK (same as YCBCR with an extra Black channel) color space.

11.27 TComponentClass Type

Unit: WebCore

```
TComponentClass = class of TComponent
```

Available In: Client and Server Applications

The TComponentClass type is used to represent a TComponent class type, and allows TComponentClass variables to store references to TComponent classes and descendants.

11.28 TComponentName Type

Unit: WebCore

```
TComponentName = type String
```

Available In: Client and Server Applications

The TComponentName type is used to represent the name of a TComponent class instance. This type is type-equivalent to a String type, but is used to distinguish the name of a component instance when used with special design-time property editors.

11.29 TCompositeOperation Type

Unit: WebUI

```
TCompositeOperation = (coSourceOver,coSourceOnTop,coSourceIn,
      coSourceOut, coDestOnTop,coDestIn,coDestOut,coDestOver,
      coLighter,coXOR,coCopy)
```

Available In: Client Applications

The TCompositeOperation enumerated type is used to specify the how pixels are combined when drawing occurs with a TCanvasElement instance.

Element	Description
coCopy	Draw the source pixel, ignoring the destination pixel.
coDestIn	Multiply the destination pixel by the opacity of the source pixel, but ignore the color of the source pixel.
coDestOnTop	Draw the source pixel underneath the destination pixel. If the source pixel is transparent, then the resulting pixel will also be transparent.
coDestOut	The destination pixel becomes transparent if the source pixel is opaque, and is not changed if the source pixel is transparent. The color of the source pixel is ignored.
coDestOver	The source pixel will appear behind the destination pixel, and shows based upon the transparency of the destination pixel.
coLighter	The colors of the source and destination pixels are added together and truncated by the maximum color value possible.
coSourceIn	Draw the source pixel, but multiply it by the opacity of the destination pixel. The color of the destination pixel is ignored, but if the destination pixel is transparent, then the resulting pixel will be also be transparent.
coSourceOnTop	Draw the source pixel on top of the destination, but multiply it by the opacity of the desination pixel. If the destination pixel is transparent, then nothing is drawn.
coSourceOut	The resulting pixel is the source pixel when the destination pixel is transparent, and a transparent pixel when the destination pixel is opaque. The color of the destination pixel is ignored.
coSourceOver	The source pixel is drawn on top of the destination pixel. If the source pixel is partially-transparent, then the destination pixel is included to draw the resulting pixel. This is the default type of composite operation for canvas drawing.
coXOR	If the source pixel is transparent, then the resulting pixel is the destination pixel. If the destination pixel is transparent, then the resulting pixel is the source pixel. If both the source and destination pixels are transparent or opaque, then the resulting pixel will be transparent.

11.30 TCompressionFormat Type

Unit: WebSrvr

```
TCompressionFormat = (cfZlib,cfGZip)
```

Available In: Server Applications

The TCompressionFormat enumerated type is used to specify the type of compression to use with the TStream Compress and Decompress methods.

Element	Description
cfGZip	Specifies that the indicated portion of the stream should be compressed using the GZip format (Deflate compression algorithm with a special GZip header).
cfZlib	Specifies that the indicated portion of the stream should be compressed using the ZLib format (Deflate compression algorithm with special ZLib header/footer).

11.31 TContent Type

Unit: WebLabels

```
TContent = type String
```

Available In: Client Applications

The TContent type is used to represent the content of a control. This type is type-equivalent to a String type, but is used to distinguish the content when used with special design-time property editors in order to allow for embedded HTML.

11.32 TContentAlignment Type

Unit: WebUI

```
TContentAlignment = (caLeft,caCenter,caRight)
```

Available In: Client Applications

The TContentAlignment enumerated type is used to specify the horizontal alignment of content in UI elements and controls.

Element	Description
caCenter	Specifies that the content will be centered.
caLeft	Specifies that the content will be left-justified.
caRight	Specifies that the content will be right-justified.

11.33 TContentDirection Type

Unit: WebUI

```
TContentDirection = (cdLeftToRight,cdRightToLeft)
```

Available In: Client Applications

The TContentDirection enumerated type is used to specify the text direction of content in UI elements and controls.

Element	Description
cdLeftToRight	Specifies that the content will be displayed/edited from left-to-right.
cdRightToLeft	Specifies that the content will be displayed/edited from right-to-left.

11.34 TContentPosition Type

Unit: WebCtrls

```
TContentPosition = (cpSpecified,cpTopLeft,cpTopCenter,cpTopRight,
       cpCenterLeft,cpCenterCenter,cpCenterRight, cpBottomLeft,
      cpBottomCenter,cpBottomRight)
```

Available In: Client Applications

The TContentPosition enumerated type is used with the TContentLayout class to specify how content elements should be positioned for UI elements and controls.

Element	Description
cpBottomCenter	Specifies that the content element should be centered horizontally at the bottom of the containing element or control.
cpBottomLeft	Specifies that the content element should be displayed at the bottom-left of the containing element or control.
cpBottomRight	Specifies that the content element should be displayed at the bottom-right of the containing element or control.
cpCenterCenter	Specifies that the content element should be centered horizontally and vertically within the containing element or control.
cpCenterLeft	Specifies that the content element should be centered vertically at the left of the containing element or control.
cpCenterRight	Specifies that the content element should be centered vertically at the right of the containing element or control.
cpSpecified	Specifies that the content element should be positioned according to the TContentLayout Left and Top properties.
cpTopCenter	Specifies that the content element should be centered horizontally at the top of the containing element or control.
cpTopLeft	Specifies that the content element should be displayed at the top-left of the containing element or control.
cpTopRight	Specifies that the content element should be displayed at the top-right of the containing element or control.

11.35 TContentSize Type

Unit: WebCtrls

```
TContentSize = (csNone,csSpecified,csContain,csCover)
```

Available In: Client Applications

The TContentSize enumerated type is used with the TContentLayout class to specify how content elements should be sized for UI elements and controls.

Element	Description
csContain	Specifies that the content element should be proportionally sized so that it fits within the bounds of the element or control.
csCover	Specifies that the content element should be proportionally sized so that the image covers the bounds of the element or control.
csNone	Specifies that the content element should not be sized and should remain its original size.
csSpecified	Specifies that the content element should be sized according to the TContentLayout Width and Height properties.

11.36 TContextMenuEvent Type

Unit: WebCtrls

```
TContextMenuEvent = function (Sender: TObject; Button: Integer;
      ShiftKey, CtrlKey, AltKey: Boolean; X,Y: Integer): Boolean of
      object
```

11.37 TControlClass Type

Unit: WebCtrls

```
TControlClass = class of TControl
```

Available In: Client Applications

The TControlClass type is used to represent a TControl class type, and allows TControlClass variables to store references to TControl classes and descendants.

11.38 TCursor Type

Unit: WebUI

```
TCursor = (crAuto,crCrossHair,crDefault,crHelp,crMove,crPointer,
      crProgress,crSizeNESW,crSizeNS,crSizeNWSE,crSizeWE, crText,
      crWait)
```

Available In: Client Applications

The TCursor enumerated type is used to specify the type of cursor that will be used for the mouse pointer when it hovers over a UI element or control.

Element	Description
crAuto	Specifies that the mouse pointer will use a cursor that is automatically determined by the web browser, based upon the type of control and its state.
crCrossHair	Specifies that the mouse pointer will use a cursor with a cross-hair (+) pattern.
crDefault	Specifies that the mouse pointer will use the default cursor for the web browser, which is usually a normal pointer.
crHelp	Specifies that the mouse pointer will use a help cursor, which is normally a pointer with a question mark (?) next to it.
crMove	Specifies that the mouse pointer will use a movement cursor, which is normally four arrows pointing north, east, south, and west.
crPointer	Specifies that the mouse pointer will use a normal pointer cursor.
crProgress	Specifies that the mouse pointer will use a cursor that represents execution in progress, which is normally a pointer with an animated progress symbol next to it.
crSizeNESW	Specifies that the mouse pointer will use a cursor that contains arrows pointing northeast and southwest.
crSizeNS	Specifies that the mouse pointer will use a cursor that contains arrows pointing north and south.
crSizeNWSE	Specifies that the mouse pointer will use a cursor that contains arrows pointing northwest and southeast. This is cursor is normally the same as the crSizeNESW cursor.
crSizeWE	Specifies that the mouse pointer will use a cursor that contains arrows pointing from west and east.
crText	Specifies that the mouse pointer will use a cursor that includes an edit caret (vertical bar).
crWait	Specifies that the mouse pointer will use a cursor that represents a wait state, which is normally an hourglass or animated progress symbol.

11.39 TDatabaseClass Type

Unit: WebData

```
TDatabaseClass = class of TDatabase
```

Available In: Client and Server Applications

The TDatabaseClass type is used to represent a TDatabase class type, and allows TDatabaseClass variables to store references to TDatabase classes and descendants.

11.40 TDatabaseErrorEvent Type

Unit: WebData

```
TDatabaseErrorEvent = procedure (Sender: TObject; const
      ErrorMsg: String) of object
```

Available In: Client and Server Applications

The TDatabaseErrorEvent type is used by the TDatabase OnCommitError and OnRollbackError events.

The Sender parameter is the TDatabase instance that triggered the event and the ErrorMsg parameter is the complete error message.

11.41 TDatabaseEvent Type

Unit: WebData

```
TDatabaseEvent = function (Sender: TObject): Boolean of object
```

Available In: Client and Server Applications

The TDatabaseEvent type is used by the TDatabase BeforeCommit and BeforeRollback events.

The Sender parameter is the TDatabase instance that triggered the event. Return True from the event to allow the applicable database functionality to continue, or False to prevent the functionality from occurring.

11.42 TDataColumnTextEvent Type

Unit: WebData

```
TDataColumnTextEvent = function (Sender: TObject; const Value:
      String): String of object
```

Available In: Client and Server Applications

The TDataColumnTextEvent type is used by the TDataColumn OnGetText and OnSetText events.

The Sender parameter is the TDataColumn instance that triggered the event and the Value parameter is the value of the column as a string (Get) or the display text being assigned to the column.

11.43 TDataRowEvent Type

Unit: WebData

```
TDataRowEvent = procedure (Sender: TObject; Column: TDataColumn)
      of object
```

Available In: Client and Server Applications

The TDataRowEvent type is used by the TDataSet OnRowChanged event.

The Sender parameter is the TDataSet instance that triggered the event and the Column parameter is the column that was changed, if applicable. If the Column parameter is nil, then the entire row has been changed, as opposed to a single column.

11.44 TDataSetErrorEvent Type

Unit: WebData

```
TDataSetErrorEvent = procedure (Sender: TObject; const ErrorMsg:
      String) of object
```

Available In: Client and Server Applications

The TDataSetErrorEvent type is used by the TDataSet OnLoadError event.

The Sender parameter is the TDataSet instance that triggered the event and the ErrorMsg parameter is the complete error message.

11.45 TDataSetEvent Type

Unit: WebData

```
TDataSetEvent = function (Sender: TObject): Boolean of object
```

Available In: Client and Server Applications

The TDataSetEvent type is used by the TDataSet BeforeOpen, BeforeClose, BeforeScroll, BeforeInsert, BeforeUpdate, BeforeSave, BeforeCancel, BeforeDelete, and BeforeLoad events.

The Sender parameter is the TDataSet instance that triggered the event. Return True from the event to allow the applicable dataset functionality to continue, or False to prevent the functionality from occurring.

11.46 TDataSetState Type

Unit: WebData

```
TDataSetState = (dsClosed,dsBrowse,dsInsert,dsUpdate)
```

Available In: Client and Server Applications

The TDataSetState enumerated type is used to indicate the State of a TDataSet instance.

Element	Description
dsBrowse	Indicates that the dataset is open and in the browse (read) state.
dsClosed	Indicates that the dataset is closed.
dsInsert	Indicates that the dataset is open and in the insert state. A dataset is put into the insert state by calling the Insert method.
dsUpdate	Indicates that the dataset is open and in the update state. A dataset is put into the update state by calling the Update method.

11.47 TDataType Type

Unit: WebData

```
TDataType = (dtUnknown,dtString,dtBoolean,dtInteger,dtFloat,
      dtDate, dtTime,dtDateTime,dtBlob)
```

Available In: Client and Server Applications

The TDataType enumerated type is used to specify the type of a TDataColumn instance in a TDataSet instance.

Element	Description
dtBlob	Specifies that the column is a BLOB (Binary Large Object) column.
dtBoolean	Specifies that the column is a boolean (True/False) column.
dtDate	Specifies that the column is a date column.
dtDateTime	Specifies that the column is a date/time column.
dtFloat	Specifies that the column is a floating-point precision numeric column.
dtInteger	Specifies that the column is an integer column.
dtString	Specifies that the column is a character string column.
dtTime	Specifies that the column is a time column.
dtUnknown	Specifies that the column is of an unknown type (do not use).

11.48 TDateTimeFormat Type

Unit: WebCore

```
TDateTimeFormat = (dtfRaw,dtfISO8601)
```

Available In: Client and Server Applications

The TDateTimeFormat enumerated type is used with TReader and TWriter classes to specify how published DateTime properties are handled when reading/writing TPersistent-descendant classes to/from JSON.

Element	Description
dtfISO8601	Date-time values are handled as ISO-8601 date-time string values.
dtfRaw	Date-time values are handled as raw numeric DateTime values (the default).

11.49 TDrawStyle Type

Unit: WebUI

```
TDrawStyle = (dsColor,dsGradient,dsPattern)
```

Available In: Client Applications

The TDrawStyle enumerated type is used to specify the drawing style for stroke and fill operations on a TCanvasElement instance.

Element	Description
dsColor	Specifies that the drawing will use a solid color. This is the default drawing style.
dsGradient	Specifies that the drawing will use a gradient.
dsPattern	Specifies that the drawing will use a pattern.

11.50 TDropDownPosition Type

Unit: WebEdits

```
TDropDownPosition = (dpRelative,dpSurfaceCenter)
```

Available In: Client Applications

The TDropDownPosition enumerated type is used with the TEditComboBox, TDateEditComboBox, adn TButtonComboBox components to specify the position of the drop-down list or calendar.

Element	Description
dpRelative	Specifies that the drop-down control will be positioned relative to the owner control. This is usually beneath the owner control, but may also be above the control is there is insufficient space in which to show the entire drop-down control.
dpSurfaceCenter	Specifies that the drop-down control will be positioned in the center of the application surface. This is especially useful for mobile applications where the browser viewport is limited in size. The application surface is represented by the Surface property of the global Application variable in the WebForms unit.

11.51 TElementClass Type

Unit: WebUI

```
TElementClass = class of TElement
```

Available In: Client Applications

The TElementClass type is used to represent a TElement class type, and allows TElementClass variables to store references to TElement classes and descendants.

11.52 TEncryptionType Type

Unit: WebSrvr

```
TEncryptionType = (etAES128,etAES192,etAES256)
```

Available In: Server Applications

The TEncryptionType enumerated type is used to specify the type of encryption to use with the TStream Encrypt and Decrypt methods.

> **Note**
> The AES (Advanced Encryption Standard, originally Rijndael) encryption algorithm is the only supported type of encryption at this time.

Element	Description
etAES128	Specifies that the indicated portion of the stream should be encrypted using the AES-128 format (block size of 128-bits with an 128-bit key).
etAES192	Specifies that the indicated portion of the stream should be encrypted using the AES-192 format (block size of 128-bits with an 192-bit key).
etAES256	Specifies that the indicated portion of the stream should be encrypted using the AES-256 format (block size of 128-bits with an 256-bit key).

11.53 TErrorEvent Type

Unit: WebRequest

```
TErrorEvent = function (Sender: TObject; const Message: String):
      Boolean of object
```

Available In: Client and Server Applications

The TErrorEvent type is used with the TApplication OnError event to trap unhandled exceptions that have bubbled up to the global Application object in a visual application.

Return True from the event to indicate to the web browser that the error was handled, or False to indicate to the web browser that the error should be handled by the browser.

11.54 TFillType Type

Unit: WebUI

```
TFillType = (ftSolid,ftGradient)
```

Available In: Client Applications

The TFillType enumerated type is used with the TFill class to specify how backgrounds should be filled for UI elements and controls.

Element	Description
ftGradient	Specifies that the background fill will be a gradient.
ftSolid	Specifies that the background fill will be a solid color (or transparent).

11.55 TFormControlClass Type

Unit: WebForms

```
TFormControlClass = class of TFormControl
```

Available In: Client Applications

The TFormControlClass type is used to represent a TFormControl class type, and allows TFormControlClass variables to store references to TFormControl classes and descendants.

11.56 TGenericFontFamily Type

Unit: WebUI

```
TGenericFontFamily = (gfSansSerif,gfSerif,gfMonospace,gfCursive,
      gfFantasy)
```

Available In: Client Applications

The TGenericFontFamily enumerated type is used with the TFont class to specify the generic font family for the fonts used in UI elements and controls. The generic font family is used as a fall-back if a specified font name is not available on the operating system that is hosting the web browser.

> **Note**
> At design-time, the IDE will automatically set the generic font family when the font name is changed for a TFont class instance, so in most cases you will never need to modify the generic font family.

Element	Description
gfCursive	The cursive generic font family, which includes script fonts.
gfFantasy	The fantasy generic font family, which includes decorative fonts.
gfMonospace	The monospace generic font family, which includes all fixed-width fonts.
gfSansSerif	The sans-serif generic font family, which includes all fonts with plain stroke endings.
gfSerif	The serif generic font family, which includes all fonts with flared or decorative stroke endings.

11.57 TGradientType Type

Unit: WebUI

```
TGradientType = (gtLinear,gtRadial)
```

Available In: Client Applications

The TGradientType enumerated type is used with the TGradient class to specify the type of gradient background to be used for UI elements and controls.

> **Note**
> This setting is only valid if the UI element or control's background is set to use a gradient fill.

Element	Description
gtLinear	Specifies that the gradient will be linear. This is the default gradient type.
gtRadial	Specifies that the gradient will be radial.

11.58 TGridColumnCellEvent Type

Unit: WebGrids

```
TGridColumnCellEvent = procedure (Sender: TObject; ACell:
      TGridCell) of object
```

Available In: Client Applications

The TGridColumnCellEvent type is used by the TGridColumn OnCellUpdate event.

The Sender parameter is the TGridColumn instance that triggered the event and the ACell parameter is the TGridCell instance being updated.

11.59 TGridColumnCompareEvent Type

Unit: WebGrids

```
TGridColumnCompareEvent = function (const L,R: String;
      CaseInsensitive: Boolean; LocaleInsensitive: Boolean): Integer
      of object
```

Available In: Client Applications

The TGridColumnCompareEvent type is used by the TGridColumn OnCompare event.

The Sender parameter is the TGridColumn instance that triggered the event, the L and R parameters are the string column values to compare, and the CaseInsensitive and LocaleInsensitive parameters indicate what type of comparison is being requested.

11.60 TGridColumnControlType Type

Unit: WebGrids

```
TGridColumnControlType = (ctNone,ctEdit,ctEditComboBox,
      ctDialogEditComboBox, ctCheckBox,ctLink,ctIcon,ctImage,
      ctDateEditComboBox, ctMultiLineEdit,ctHTML)
```

Available In: Client Applications

The TGridColumnControlType enumerated type is used with the TGridColumn class to specify the type of control to be used when displaying or editing a grid column.

Element	Description
ctCheckBox	Specifies that the grid column will use a checkbox control for editing its cells.
ctDateEditComboBox	Specifies that the grid column will use a date combo box control for editing its cells.
ctDialogEditComboBox	Specifies that the grid column will use a dialog combo box control for editing its cells.
ctEdit	Specifies that the grid column will use a single-line edit control for editing its cells.
ctEditComboBox	Specifies that the grid column will use an edit combo box control for editing its cells.
ctHTML	Specifies that the grid column will display its cells as HTML.
ctIcon	Specifies that the grid column will display its cells as icons. The content of the cells specifies the name of the icon to display.
ctImage	Specifies that the grid column will display its cells as images. The content of the cells specifies the URL of the image to display.
ctLink	Specifies that the grid column will display its cells as links.
ctMultiLineEdit	Specifies that the grid column will use a multi-line edit control for editing its cells.
ctNone	Specifies that the grid column will not use any special controls for editing/displaying. This is the default value, and using this value will result in the grid column not being editable.

11.61 TGridColumnUpdateEvent Type

Unit: WebGrids

```
TGridColumnUpdateEvent = procedure (Sender: TObject; ARowIndex:
      Integer) of object
```

Available In: Client Applications

The TGridColumnUpdateEvent type is used by the TGridColumn OnUpdate event.

The Sender parameter is the TGridColumn instance that triggered the event and the RowIndex parameter is the index of the row that was updated in the list of rows in the TGrid Rows property.

11.62 TGridHeaderClickEvent Type

Unit: WebGrids

```
TGridHeaderClickEvent = function (Sender: TObject): Boolean of
      object
```

Available In: Client Applications

The TGridHeaderClickEvent type is used by the TGridColumn OnHeaderClick event.

The Sender parameter is the TGridColumn instance that triggered the event.

11.63 THandleRequestEvent Type

Unit: WebRequest

```
THandleRequestEvent = procedure (Sender: TObject; Request:
      TWebServerRequest) of object
```

Available In: Server Applications

The THandleRequestEvent type is used by the TRequestHandler OnHandleRequest event to handle incoming web server requests.

The Sender parameter is the TRequestHandler instance that triggered the event and the Request parameter is the TWebServerRequest instance that represents the incoming web server request.

11.64 THashType Type

Unit: WebSrvr

```
THashType = (htMD5,htSHA1,htSHA256,htSHA512)
```

Available In: Server Applications

The THashType enumerated type is used to specify the type of hash to compute with the TStream ComputeHash method.

Element	Description
htMD5	Specifies that the computed hash will be a 128-bit MD-5 (Message Digest) hash. **Warning** MD-5 hashes are cryptographic hashes, but are **not** secure and should only be used for purposes such as computing checksums for data.
htSHA1	Specifies that the computed hash will be a 160-bit SHA-1 (Secure Hash Algorithm) hash. **Warning** SHA-1 hashes are cryptographic hashes, but are **not** secure and should only be used for purposes such as computing checksums for data.
htSHA256	Specifies that the computed hash will be a 256-bit SHA-2 (Secure Hash Algorithm) hash.
htSHA512	Specifies that the computed hash will be a 512-bit SHA-2 (Secure Hash Algorithm) hash.

11.65 THTMLFormEncoding Type

Unit: WebUI

```
THTMLFormEncoding = (feMultiPartFormData,feURLEncoded,
      feTextPlain)
```

Available In: Client Applications

The THTMLFormEncoding enumerated type is used with the THTMLForm and TFormElement classes to specify the type of encoding to use for any form values when the HTML form is submitted to the web server.

Element	Description
feMultiPartFormData	Specifies that the form should encode the form values using the multipart/form-data MIME type. This is the default type of encoding, and should always be used when uploading files in addition to text values.
feTextPlain	Specifies that the form should encode the form values using the text/plain MIME type. The only encoding that will take place is that spaces will be converted into "+" characters.
feURLEncoded	Specifies that the form should encode the form values using the application/x-www-form-urlencoded MIME type. This encoding will cause spaces to be converted into "+" characters and any special characters to be converted into ASCII hex values prefixed with the "%" character.

11.66 THTMLFormMethod Type

Unit: WebUI

```
THTMLFormMethod = (fmGet,fmPost,fmHead,fmPut,fmDelete)
```

Available In: Client Applications

The THTMLFormMethod enumerated type is used to specify the HTTP method to use for the THTMLForm component when the Submit method is called to submit the form values to the web server.

Element	Description
fmDelete	Specifies that the HTTP DELETE method will be used to submit the HTML form values.
fmGet	Specifies that the HTTP GET method will be used to submit the HTML form values.
fmHead	Specifies that the HTTP HEAD method will be used to submit the HTML form values.
fmPost	Specifies that the HTTP POST method will be used to submit the HTML form values. This is the default method.
fmPut	Specifies that the HTTP PUT method will be used to submit the HTML form values.

11.67 THTTPMethod Type

Unit: WebSrvr

```
THTTPMethod = (hmUnknown,hmGet,hmHead,hmPost,hmPut,hmDelete,
      hmOptions, hmPatch)
```

Available In: Server Applications

The THTTPMethod enumerated type is used by the THTTPServerRequest class to indicate the HTTP method for a web server request.

Element	Description
hmDelete	Indicates that the request is an HTTP DELETE request.
hmGet	Indicates that the request is an HTTP GET request.
hmHead	Indicates that the request is an HTTP HEAD request.
hmOptions	Indicates that the request is an HTTP OPTIONS request. **Note** This type of request is always handled internally by the web server and will never appear in a request routed to a server application or native web server module.
hmPatch	Indicates that the request is an HTTP PATCH request.
hmPost	Indicates that the request is an HTTP POST request.
hmPut	Indicates that the request is an HTTP PUT request.
hmUnknown	Indicates that the type of the HTTP request is unknown.

11.68 THTTPResponseType Type

Unit: WebSrvr

```
THTTPResponseType = (hrText,hrStream)
```

Available In: Server Applications

The THTTPResponseType enumerated type is used by the THTTPRequest class to indicate how the response of an HTTP request should be handled.

Element	Description
hrStream	Specifies that the response should be stored in a stream and made accessible via the THTTPRequest ResponseStream property.
hrText	Specifies that the response should be stored as text and made accessible via the THTTPRequest ResponseText property.

11.69 TImageFormat Type

Unit: WebSrvr

```
TImageFormat = (ifUnknown,ifBMP,ifJPEG,ifGIF,ifPNG,ifTIFF)
```

Available In: Server Applications

The TImageFormat enumerated type is used to identity the format of the image loaded into a TRasterImage instance, or to specify the format of the image being saved from a TRasterImage instance using the TRasterImage SaveToFile or SaveToStream methods.

Element	Description
ifBMP	Indicates that the raster image is in the BMP (Bitmap) format.
ifGIF	Indicates that the raster image is in the GIF (Graphics Interchange Format) format.
ifJPEG	Indicates that the raster image is in the JPEG (Joint Photographic Experts Group) format.
ifPNG	Indicates that the raster image is in the PNG (Portable Network Graphics) format.
ifTIFF	Indicates that the raster image is in the TIFF (Tagged Image File Format) format.
ifUnknown	Indicates that the raster image is in an unknown format.

11.70 TIntegerArray Type

Unit: WebCore

```
TIntegerArray = array of Integer
```

Available In: Client and Server Applications

The TIntegerArray type is used in classes such as the TSet class to represent an array of Integer values.

11.71 TInterfaceAnimationEvent Type

Unit: WebUI

```
TInterfaceAnimationEvent = procedure (ACurrentTime: Double) of
      object
```

Available In: Client Applications

The TInterfaceAnimationEvent type is used by the TInterfaceManager BeginAnimation and ContinueAnimation methods to specify an animation event handler to be used for animating the frames of an animation.

The AStartTime parameter is the time, in milliseconds, when the animation frame was initiated. This value can be used by the event handler to reliably calculate animation effects based upon elapsed time.

11.72 TInterfaceControllerClass Type

Unit: WebUI

```
TInterfaceControllerClass = class of TInterfaceController
```

Available In: Client Applications

The TInterfaceControllerClass type is used to represent a TInterfaceController class type, and allows TInterfaceControllerClass variables to store references to TInterfaceController classes and descendants.

11.73 TInterfaceErrorEvent Type

Unit: WebUI

```
TInterfaceErrorEvent = function (const Message: String): Boolean
      of object
```

Available In: Client Applications

The TInterfaceErrorEvent type is used by the TInterfaceManager class to specify a global error handler for an application.

> **Note**
> The global TApplication instance automatically created for visual applications assigns an event handler for interface manager errors, so you should not normally need to deal with this event type.

11.74 TInterfaceIdleEvent Type

Unit: WebUI

```
TInterfaceIdleEvent = procedure of object
```

Available In: Client Applications

The TInterfaceIdleEvent type is used by the TInterfaceManager OnIdle event to handle user inactivity timeouts.

11.75 TInterfaceTimeoutEvent Type

Unit: WebUI

```
TInterfaceTimeoutEvent = procedure of object
```

Available In: Client Applications

The TInterfaceTimeoutEvent type is used by the TInterfaceManager CreateTimeout method to create a timeout. A timeout waits N milliseconds, and then executes the specified event handler.

11.76 TInterfaceTimerEvent Type

Unit: WebUI

```
TInterfaceTimerEvent = procedure of object
```

Available In: Client Applications

The TInterfaceTimerEvent type is used by the TInterfaceManager CreateTimer method to create a timer. A timer executes the specified event handler every N milliseconds.

11.77 TInterfaceViewportResizeEvent Type

Unit: WebUI

```
TInterfaceViewportResizeEvent = procedure of object
```

Available In: Client Applications

The TInterfaceViewportResizeEvent type is used by the TInterfaceManager to indicate when the browser viewport has been resized.

11.78 TInterfaceViewportScrollEvent Type

Unit: WebUI

```
TInterfaceViewportScrollEvent = procedure of object
```

Available In: Client Applications

The TInterfaceViewportScrollEvent type is used by the TInterfaceManager to indicate when the browser viewport has been scrolled in any direction.

11.79 TKeyDownEvent Type

Unit: WebCtrls

```
TKeyDownEvent = function (Sender: TObject; Key: Integer;
      ShiftKey, CtrlKey, AltKey: Boolean): Boolean of object
```

Available In: Client Applications

The TKeyDownEvent type is a common event type that is used by controls to provide notification that a key is being pressed.

The Sender parameter represents the class instance that triggered the event. The Key parameter represents the ordinal key code of the key pressed, and the ShiftKey, CtrlKey, AltKey parameters represent whether the Shift, Control, and/or Alt keys were also pressed.

The Key value may represent a virtual key code. Certain keys, such as the Insert, Delete, and navigation keys, use virtual key codes since these keys do not correspond to an ordinal Unicode character value. The following virtual key code constants are defined in the WebUI unit for your convenience:

```
VK_BACK = 8;
VK_TAB = 9;
VK_RETURN = 13;
VK_ESCAPE = 27;
VK_SPACE = 32;
VK_PRIOR = 33;
VK_NEXT = 34;
VK_END = 35;
VK_HOME = 36;
VK_LEFT = 37;
VK_UP = 38;
VK_RIGHT = 39;
VK_DOWN = 40;
VK_INSERT = 45;
VK_DELETE = 46;
```

To not allow the keystroke, return False as the result to any event handler attached to this event. To allow the keystroke, return True.

11.80 TKeyPressEvent Type

Unit: WebCtrls

```
TKeyPressEvent = function (Sender: TObject; Key: Char; ShiftKey,
      CtrlKey, AltKey: Boolean): Boolean of object
```

Available In: Client and Server Applications

The TKeyPressEvent type is a common event type that is used by controls to provide notification that a key has been pressed.

The Sender parameter represents the class instance that triggered the event. The Key parameter represents the character key code of the key pressed, and the ShiftKey, CtrlKey, AltKey parameters represent whether the Shift, Control, and/or Alt keys were also pressed.

Warning
Certain keys, such as the Insert, Delete, and navigation keys, use virtual key codes since these keys do not correspond to a Unicode character value. Only use this event to handle keystrokes that correspond to Unicode character values. Use the OnKeyDown event to handle keystrokes that use virtual key codes.

To not allow the keystroke, return False as the result to any event handler attached to this event. To allow the keystroke, return True.

11.81 TKeyUpEvent Type

Unit: WebCtrls

```
TKeyUpEvent = procedure (Sender: TObject; Key: Integer; ShiftKey,
       CtrlKey, AltKey: Boolean) of object
```

Available In: Client and Server Applications

The TKeyUpEvent type is a common event type that is used by controls to provide notification that a key is being released.

The Sender parameter represents the class instance that triggered the event. The Key parameter represents the ordinal key code of the key pressed, and the ShiftKey, CtrlKey, AltKey parameters represent whether the Shift, Control, and/or Alt keys were also pressed.

The Key value may represent a virtual key code. Certain keys, such as the Insert, Delete, and navigation keys, use virtual key codes since these keys do not correspond to an ordinal Unicode character value. The following virtual key code constants are defined in the WebUI unit for your convenience:

```
VK_BACK = 8;
VK_TAB = 9;
VK_RETURN = 13;
VK_ESCAPE = 27;
VK_SPACE = 32;
VK_PRIOR = 33;
VK_NEXT = 34;
VK_END = 35;
VK_HOME = 36;
VK_LEFT = 37;
VK_UP = 38;
VK_RIGHT = 39;
VK_DOWN = 40;
VK_INSERT = 45;
VK_DELETE = 46;
```

11.82 TLayoutConsumption Type

Unit: WebUI

```
TLayoutConsumption = (lcNone,lcTopLeft,lcTop,lcTopRight,lcLeft,
      lcRight, lcBottomLeft,lcBottom,lcBottomRight)
```

Available In: Client and Server Applications

The TLayoutConsumption enumerated type is used with the TLayout class to specify how a UI element or control consumes space in the layout rectangle.

Element	Description
lcBottom	Specifies that the element or control will consume space towards the bottom of the layout rectangle.
lcBottomLeft	Specifies that the element or control will consume space towards the bottom-left corner of the layout rectangle.
lcBottomRight	Specifies that the element or control will consume space towards the bottom-right corner of the layout rectangle.
lcLeft	Specifies that the element or control will consume space towards the left side of the layout rectangle.
lcNone	Specifies that the element or control will not consume any space in the layout rectangle. This is the default value.
lcRight	Specifies that the element or control will consume space towards the right side of the layout rectangle.
lcTop	Specifies that the element or control will consume space towards the top of the layout rectangle.
lcTopLeft	Specifies that the element or control will consume space towards the top-left corner of the layout rectangle.
lcTopRight	Specifies that the element or control will consume space towards the top-right corner of the layout rectangle.

11.83 TLayoutOverflow Type

Unit: WebUI

```
TLayoutOverflow = (loNone,loTop,loLeft,loRight,loBottom)
```

Available In: Client and Server Applications

The TLayoutOverflow enumerated type is used with the TLayout class to specify how a UI element or control handles situations where it won't fit within the bounds of the current layout rectangle.

Element	Description
loBottom	Specifies that the prior element or control's consumption will be temporarily reset so that space is consumed towards the bottom of the current layout rectangle.
loLeft	Specifies that the prior element or control's consumption will be temporarily reset so that space is consumed towards the left side of the current layout rectangle.
loNone	Specifies that the prior element or control's consumption will not be temporarily reset.
loRight	Specifies that the prior element or control's consumption will be temporarily reset so that space is consumed towards the right side of the current layout rectangle.
loTop	Specifies that the prior element or control's consumption will be temporarily reset so that space is consumed towards the top of the current layout rectangle.

11.84 TLayoutPosition Type

Unit: WebUI

```
TLayoutPosition = (lpNone,lpTopLeft,lpTop,lpTopCenter,lpTopRight,
       lpLeft,lpLeftCenter,lpCenter,lpRight,lpRightCenter,
      lpBottomLeft,lpBottom,lpBottomCenter,lpBottomRight)
```

Available In: Client and Server Applications

The TLayoutPosition enumerated type is used with the TLayout class to specify how a UI element or control is positioned in the layout rectangle.

Element	Description
lpBottom	Specifies that the element or control will be positioned at the bottom of the layout rectangle.
lpBottomCenter	Specifies that the element or control will be centered horizontally at the bottom of the layout rectangle.
lpBottomLeft	Specifies that the element or control will be positioned in the bottom left corner of the layout rectangle.
lpBottomRight	Specifies that the element or control will be positioned in the bottom right corner of the layout rectangle.
lpCenter	Specifies that the element or control will be centered horizontally and vertically within the layout rectangle.
lpLeft	Specifies that the element or control will be positioned on the left side of the layout rectangle.
lpLeftCenter	Specifies that the element or control will be centered vertically on the left side of the layout rectangle.
lpNone	Specifies that the element or control will not be positioned by the control's layout management, and will instead be positioned according to its assigned Left and Top property values. This is the default value.
lpRight	Specifies that the element or control will be positioned on the right side of the layout rectangle.
lpRightCenter	Specifies that the element or control will be centered vertically on the right side of the layout rectangle.
lpTop	Specifies that the element or control will be positioned at the top of the layout rectangle.
lpTopCenter	Specifies that the element or control will be centered horizontally at the top of the layout rectangle.
lpTopLeft	Specifies that the element or control will be positioned in the top left corner of the layout rectangle.
lpTopRight	Specifies that the element or control will be positioned in the top right corner of the layout rectangle.

11.85 TLayoutStretch Type

Unit: WebUI

```
TLayoutStretch = (lsNone,lsTopLeft,lsTop,lsTopRight,lsLeft,
      lsRight, lsBottomLeft,lsBottom,lsBottomRight)
```

Available In: Client and Server Applications

The TLayoutStretch enumerated type is used with the TLayout class to specify how a UI element or control is stretched to fill parts of the layout rectangle.

Element	Description
lsBottom	Specifies that the element or control will stretch to the bottom of the layout rectangle.
lsBottomLeft	Specifies that the element or control will stretch to the bottom-left corner of the layout rectangle.
lsBottomRight	Specifies that the element or control will stretch to the bottom-right corner of the layout rectangle.
lsLeft	Specifies that the element or control will stretch to the left side of the layout rectangle.
lsNone	Specifies that the element or control will not be stretched. This is the default value.
lsRight	Specifies that the element or control will stretch to the right side of the layout rectangle.
lsTop	Specifies that the element or control will stretch to the top of the layout rectangle.
lsTopLeft	Specifies that the element or control will stretch to the top-left corner of the layout rectangle.
lsTopRight	Specifies that the element or control will stretch to the top-right corner of the layout rectangle.

11.86 TLineCapStyle Type

Unit: WebUI

```
TLineCapStyle = (csButt,csRound,csSquare)
```

Available In: Client and Server Applications

The TLineCapStyle enumerated type is used to specify the how lines are terminated on a TCanvasElement instance.

Element	Description
csButt	Specifies that lines should have no cap. The end of the lines will be straight and perpendicular to the direction of the lines, and the lines are not extended beyond their endpoint.
csRound	Specifies that lines should be capped with a semicircle whose diameter is equal to the width of the line. This semicircle extends beyond the end of the line by one half of the width of the line.
csSquare	Specifies that lines should be capped with a rectangle. This is just like the csButt member, but the lines are extended by half of their width past their endpoint.

11.87 TLineJoinStyle Type

Unit: WebUI

```
TLineJoinStyle = (jsMiter,jsBevel,jsRound)
```

Available In: Client and Server Applications

The TLineJoinStyle enumerated type is used to specify how lines are drawn when they intersect on a TCanvasElement instance.

Element	Description
jsBevel	Specifies that the outside edges of the intersecting lines are joined with a filled triangle.
jsMiter	Specifies that the outside edges of the intersecting lines are extended until they meet.
jsRound	Specifies that the outside edges of the intersecting lines are joined with a filled arc whose diameter is equal to the width of the line.

11.88 TLocationError Type

Unit: WebComps

```
TLocationError = (leNone,lePermissionDenied,leUnavailable,
      leTimeout)
```

Available In: Client and Server Applications

The TLocationError enumerated type is used with the TLocationServices OnLocationError event to determine the reason why the current location cannot be obtained from the machine or device using the host web browser.

Element	Description
leNone	The location information hasn't been obtained yet, or was obtained without issue.
lePermissionDenied	The location information could not be obtained because the user did not grant permission for the current application to access the location information for the machine or device.
leTimeout	The location information could not be obtained for the machine or device within the number of milliseconds specified in the TLocationServices Timeout property.
leUnavailable	The location information could not be obtained because location services are not available for the machine or device. **Note** This error condition can occur when the current application was loaded in a non-secure context (http), while the host web browser only allows location services to be accessed from a secure context (https).

11.89 TMailerEvent Type

Unit: WebMail

```
TMailerEvent = procedure (Mailer: TMailer) of object
```

Available In: Server Applications

11.90 TMailSecurity Type

Unit: WebMail

```
TMailSecurity = (msNone,msUpgradeToSecure,msSecure)
```

Available In: Server Applications

The TMailSecurity enumerated type is used to identity the required level of security when sending email using an TMailer instance in a server application.

Element	Description
msNone	Specifies that the entire mail transaction with the mail server should be performed with no encryption at all.
msSecure	Specifies that encryption is required for the entire mail transaction with the mail server, including the initial handshake.
msUpgradeToSecure	Specifies that the initial handshake portion of the mail transaction with the mail server can be performed without encryption, but that the rest of the mail transaction will require encryption.

11.91 TMapControlPosition Type

Unit: WebMaps

```
TMapControlPosition = (cpBottomCenter,cpBottomLeft,cpBottomRight,
       cpLeftBottom,cpLeftCenter,cpLeftTop, cpRightBottom,
      cpRightCenter,cpRightTop, cpTopCenter,cpTopLeft,cpTopRight)
```

Available In: Client and Server Applications

The TMapControlPosition enumerated type is used with the TMap control to specify how various map controls (map type, overview map, pan, rotate, street view, zoom) are positioned over the map.

Element	Description
cpBottomCenter	The control is positioned at the bottom of the map, in the center.
cpBottomLeft	The control is positioned at the bottom of the map, on the left.
cpBottomRight	The control is positioned at the bottom of the map, on the right.
cpLeftBottom	The control is positioned on the left side of the map, at the bottom.
cpLeftCenter	The control is positioned on the left side of the map, in the center.
cpLeftTop	The control is positioned on the left side of the map, at the top.
cpRightBottom	The control is positioned on the right side of the map, at the bottom.
cpRightCenter	The control is positioned on the right side of the map, in the center.
cpRightTop	The control is positioned on the right side of the map, at the top.
cpTopCenter	The control is positioned at the top of the map, in the center.
cpTopLeft	The control is positioned at the top of the map, on the left.
cpTopRight	The control is positioned at the top of the map, on the right.

11.92 TMapTilt Type

Unit: WebMaps

```
TMapTilt = (mt0Degrees,mt45Degrees)
```

Available In: Client and Server Applications

The TMapTilt enumerated type is used with the TMapOptions class to specify the viewing angle of the map.

Element	Description
mt0Degrees	Specifies a viewing angle of 0 degrees (the default).
mt45Degrees	Specifies a viewing angle of 45 degrees.

11.93 TMapType Type

Unit: WebMaps

```
TMapType = (mtHybrid,mtRoadmap,mtSatellite,mtTerrain)
```

Available In: Client and Server Applications

The TMapType enumerated type is used with the TMapOptions class to specify the type of map to show.

Element	Description
mtHybrid	This map type displays a transparent layer of major streets on satellite images.
mtRoadmap	This map type displays a normal street map.
mtSatellite	This map type displays satellite images.
mtTerrain	This map type displays maps with physical features such as terrain and vegetation.

11.94 TMapTypeControlStyle Type

Unit: WebMaps

```
TMapTypeControlStyle = (tcDefault,tcDropDownMenu,
      tcHorizontalBar)
```

Available In: Client and Server Applications

The TMapTypeControlStyle enumerated type is used with the TMapTypeControlOptions class to specify the style of the map type control.

Element	Description
tcDefault	The map type control will be the default control style (a horizontal menu bar).
tcDropDownMenu	The map type control will be a drop-down menu.
tcHorizontalBar	The map type control will be a horizontal menu bar.

11.95 TMediaNetworkState Type

Unit: WebUI

```
TMediaNetworkState = (mnsEmpty,mnsIdle,mnsLoading,mnsNoSource)
```

Available In: Client and Server Applications

The TMediaNetworkState enumerated type is used with the TAudio, TVideo, and TMediaElement classes to determine the current network state of a media UI element or control.

Element	Description
mnsEmpty	The media element or control has not started using the network. This is the case right after a new media URL has been specified for the media element or control.
mnsIdle	The media element or control is not currently using the network.
mnsLoading	The media element or control is currently using the network to load media data.
mnsNoSource	The media element or control is not currently using the network because the media URL specified as a source cannot be played by the media element or control.

11.96 TMediaPreload Type

Unit: WebUI

```
TMediaPreload = (mplNone,mplMetaData,mplAuto)
```

Available In: Client and Server Applications

The TMediaPreload enumerated type is used with the TAudio, TVideo, and TMediaElement classes to specify how much media data can be pre-loaded by the media UI element or control.

Element	Description
mplAuto	All of the media data can be pre-loaded.
mplMetaData	Only the media metadata can be pre-loaded.
mplNone	None of the media data can be pre-loaded.

11.97 TMediaReadyState Type

Unit: WebUI

```
TMediaReadyState = (mrsNothing,mrsMetadata,mrsCurrentData,
      mrsFutureData,mrsEnoughData)
```

Available In: Client and Server Applications

The TMediaReadyState enumerated type is used with the TAudio, TVideo, and TMediaElement classes to determine the current playback-readiness of the media control.

Element	Description
mrsCurrentData	Media data has been loaded for the current playback position, but nothing more. This state normally occurs at the end of media playback.
mrsEnoughData	Enough media data has been loaded that the media control should be able to play until the end of the media without pausing.
mrsFutureData	Enough media data has been loaded to begin playing, but the media control will most likely have to pause at some later point to load more data.
mrsMetadata	The media metadata has been loaded, so statistics such as the duration of the media will be available, but no media data has been loaded yet.
mrsNothing	No media data has been loaded (including metadata).

11.98 TModalResult Type

Unit: WebCtrls

```
TModalResult = (mrNone,mrOk,mrCancel,mrAbort,mrRetry, mrIgnore,
      mrYes,mrNo,mrAll,mrNoToAll, mrYesToAll,mrClose)
```

Available In: Client and Server Applications

The TModalResult enumerated type is used with modal forms and dialogs to indicate which action the user selected to close the modal form or dialog.

Element	Description
mrAbort	Indicates that the Abort button was clicked or selected.
mrAll	Indicates that the All button was clicked or selected.
mrCancel	Indicates that the Cancel button was clicked or selected.
mrClose	Indicates that the Close button was clicked or selected.
mrIgnore	Indicates that the Ignore button was clicked or selected.
mrNo	Indicates that the No button was clicked or selected.
mrNone	Indicates that no button was clicked or selected.
mrNoToAll	Indicates that the No to All button was clicked or selected.
mrOk	Indicates that the Ok button was clicked or selected.
mrRetry	Indicates that the Retry button was clicked or selected.
mrYes	Indicates that the Yes button was clicked or selected.
mrYesToAll	Indicates that the Yes to All button was clicked or selected.

11.99 TMouseDownEvent Type

Unit: WebCtrls

```
TMouseDownEvent = procedure (Sender: TObject; Button: Integer;
      ShiftKey, CtrlKey, AltKey: Boolean; X,Y: Integer) of object
```

Available In: Client and Server Applications

The TMouseDownEvent type is a common event type that is used by controls to provide notification that a mouse button is being pressed.

The Sender parameter represents the class instance that triggered the event. The Button parameter represents the ordinal button code of the mouse button pressed, and the ShiftKey, CtrlKey, AltKey parameters represent whether the Shift, Control, and/or Alt keys were also pressed. The X and Y parameters indicate the horizontal and vertical position of the mouse pointer, in pixels, relative to the bounds of the control that triggered the event.

The ordinal mouse button values are defined in the WebUI unit, and are as follows:

```
MB_NONE = 0;
MB_LEFT = 1;
MB_MIDDLE = 2;
MB_RIGHT = 3;
```

11.100 TMouseMoveEvent Type

Unit: WebCtrls

```
TMouseMoveEvent = procedure (Sender: TObject; ShiftKey, CtrlKey,
      AltKey: Boolean; X,Y: Integer) of object
```

Available In: Client and Server Applications

The TMouseMoveEvent type is a common event type that is used by controls to provide notification that the mouse pointer is being moved over the control.

The Sender parameter represents the class instance that triggered the event. The ShiftKey, CtrlKey, AltKey parameters represent whether the Shift, Control, and/or Alt keys were also pressed. The X and Y parameters indicate the horizontal and vertical position of the mouse pointer, in pixels, relative to the bounds of the control that triggered the event.

The ordinal mouse button values are defined in the WebUI unit, and are as follows:

```
MB_NONE = 0;
MB_LEFT = 1;
MB_MIDDLE = 2;
MB_RIGHT = 3;
```

11.101 TMouseUpEvent Type

Unit: WebCtrls

```
TMouseUpEvent = procedure (Sender: TObject; Button: Integer;
      ShiftKey, CtrlKey, AltKey: Boolean; X,Y: Integer) of object
```

Available In: Client and Server Applications

The TMouseUpEvent type is a common event type that is used by controls to provide notification that a mouse button is being released.

The Sender parameter represents the class instance that triggered the event. The Button parameter represents the ordinal button code of the mouse button pressed, and the ShiftKey, CtrlKey, AltKey parameters represent whether the Shift, Control, and/or Alt keys were also pressed. The X and Y parameters indicate the horizontal and vertical position of the mouse pointer, in pixels, relative to the bounds of the control that triggered the event.

The ordinal mouse button values are defined in the WebUI unit, and are as follows:

```
MB_NONE = 0;
MB_LEFT = 1;
MB_MIDDLE = 2;
MB_RIGHT = 3;
```

11.102 TMouseWheelEvent Type

Unit: WebCtrls

```
TMouseWheelEvent = function (Sender: TObject; WheelDelta:
      Integer; ShiftKey, CtrlKey, AltKey: Boolean; X,Y: Integer):
      Boolean of object
```

Available In: Client and Server Applications

The TMouseWheelEvent type is a common event type that is used by controls to provide notification that the mouse wheel is being rotated.

The Sender parameter represents the class instance that triggered the event. The WheelDelta parameter represents the amount, in pixels, that the mouse wheel rotation represents, and the ShiftKey, CtrlKey, AltKey parameters represent whether the Shift, Control, and/or Alt keys were also pressed. The X and Y parameters indicate the horizontal and vertical position of the mouse pointer, in pixels, relative to the bounds of the control that triggered the event.

Note
The WheelDelta parameter can be positive or negative, depending upon the configuration of the mouse and the direction in which the mouse wheel was rotated.

11.103 TMsgDlgBtn Type

Unit: WebForms

```
TMsgDlgBtn = (mbNone,mbOk,mbCancel,mbAbort,mbRetry,mbIgnore,
      mbYes,mbNo,mbAll,mbNoToAll,mbYesToAll,mbClose)
```

Available In: Client and Server Applications

The TMsgDlgBtn enumerated type is used to specify the message dialog buttons to display in the MessageDlg procedure.

Element	Description
mbAbort	Specifies that the button is an Abort button.
mbAll	Specifies that the button is an All button.
mbCancel	Specifies that the button is a Cancel button.
mbClose	Specifies that the button is a Close button.
mbIgnore	Specifies that the button is an Ignore button.
mbNo	Specifies that the button is a No button.
mbNone	Not used.
mbNoToAll	Specifies that the button is a No to All button.
mbOk	Specifies that the button is an Ok button.
mbRetry	Specifies that the button is a Retry button.
mbYes	Specifies that the button is a Yes button.
mbYesToAll	Specifies that the button is a Yes to All button.

11.104 TMsgDlgBtns Type

Unit: WebForms

```
TMsgDlgBtns = array of TMsgDlgBtn
```

Available In: Client and Server Applications

The TMsgDlgBtns type is simply a type definition for an array of TMsgDlgBtn enumerated values, and is used in the MessageDlg procedure.

11.105 TMsgDlgResultEvent Type

Unit: WebForms

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

Available In: Client and Server Applications

The TMsgDlgResultEvent event type is used with the MessageDlg procedure to pass an event handler to the procedure that is called when the modal message dialog is closed by the user.

The DlgResult parameter indicates the button that the user clicked or selected, if any, to close the message dialog.

11.106 TMsgDlgType Type

Unit: WebForms

```
TMsgDlgType = (mtWarning,mtError,mtInformation,mtConfirmation,
      mtCustom)
```

Available In: Client and Server Applications

The TMsgDlgType enumerated type is used with the MessageDlg procedure to specify what type of message dialog should be displayed.

Element	Description
mtConfirmation	Specifies that the message dialog will be a confirmation dialog, and an applicable icon will be displayed on the message dialog to reflect this.
mtCustom	Specifies that the message dialog will be a custom dialog, and an applicable icon will be displayed on the message dialog to reflect this.
mtError	Specifies that the message dialog will be an error dialog, and an applicable icon will be displayed on the message dialog to reflect this.
mtInformation	Specifies that the message dialog will be an informational dialog, and an applicable icon will be displayed on the message dialog to reflect this.
mtWarning	Specifies that the message dialog will be a warning dialog, and an applicable icon will be displayed on the message dialog to reflect this.

11.107 TNotifyEvent Type

Unit: WebCore

```
TNotifyEvent = procedure (Sender: TObject) of object
```

Available In: Client and Server Applications

The TNotifyEvent type is a common event type that is used in any situation where a simple notification mechanism is required, such as the OnClick event. The Sender parameter is the class instance that triggered the event.

11.108 TObjectsArray Type

Unit: WebCore

```
TObjectsArray = array of TObject
```

Available In: Client and Server Applications

The TObjectsArray type is used in classes like the TObjectList class to represent an array of TObject instances.

11.109 TOverflowType Type

Unit: WebUI

```
TOverflowType = (otHidden,otAuto,otScroll)
```

Available In: Client and Server Applications

The TOverflowType type is used by the TViewport component to specify whether or not horizontal and/or vertical scrollbars should be shown when the application surface size exceeds the browser viewport size.

Element	Description
otAuto	A scrollbar is shown if the application surface exceeds the browser viewport size, otherwise no scrollbar is shown.
otHidden	No scrollbar is shown, even if the application surface exceeds the browser viewport size This is the default value.
otScroll	A scrollbar is always shown, even if the application surface does not exceed the browser viewport size.

11.110 TPageChangeEvent Type

Unit: WebPages

```
TPageChangeEvent = function (Sender: TObject; NewPage: TPage):
      Boolean of object
```

Available In: Client and Server Applications

The TPageChangeEvent event type is used by the TPagePanel OnPageChange event to indicate when the active page changes in the control.

11.111 TPatternRepeatStyle Type

Unit: WebUI

```
TPatternRepeatStyle = (psNone,psHorizontal,psVertical,psBoth)
```

Available In: Client and Server Applications

The TPatternRepeatStyle enumerated type is used to specify how a pattern should be tiled on a TCanvasElement instance.

Element	Description
psBoth	Specifies that the pattern will be tiled both horizontally and vertically.
psHorizontal	Specifies that the pattern will be tiled horizontally.
psNone	Specifies that the pattern will not be tiled at all. This is the default value.
psVertical	Specifies that the pattern will be tiled vertically.

11.112 TPixelFormat Type

Unit: WebSrvr

```
TPixelFormat = (pfUnknown,pf24BppRGB,pf32BppRGB,pf32BppARGB,
      pf32BppPARGB, pf48BppRGB,pf64BppARGB,pf64BppPARGB,pf32BppCMYK)
```

Available In: Client and Server Applications

The TPixelFormat enumerated type is used to identity the pixel format of the image in a TRasterImage instance.

Element	Description
pf24BppRGB	The pixel format of the image is 24-bit RGB (Red, Green and Blue channels).
pf32BppARGB	The pixel format of the image is 32-bit ARGB (Alpha, Red, Green and Blue channels).
pf32BppCMYK	The pixel format of the image is 32-bit CMYK (Cyan, Magenta, Yellow and Black channels).
pf32BppPARGB	The pixel format of the image is 32-bit ARGB (Alpha, Red, Green and Blue channels where Red, Green, and Blue channels are premultiplied with Alpha channel).
pf32BppRGB	The pixel format of the image is 32-bit RGB (Red, Green and Blue channels, with an unused channel).
pf48BppRGB	The pixel format of the image is 48-bit RGB (Red, Green and Blue channels are 16-bit instead of 8-bit).
pf64BppARGB	The pixel format of the image is 64-bit RGB (Alpha, Red, Green and Blue channels are 16-bit instead of 8-bit).
pf64BppPARGB	The pixel format of the image is 64-bit RGB (Alpha, Red, Green and Blue channels are 16-bit instead of 8-bit and Red, Green, and Blue channels are premultiplied with Alpha channel).
pfUnknown	The pixel format is unknown.

11.113 TRequestHandlerClass Type

Unit: WebRequest

```
TRequestHandlerClass = class of TRequestHandler
```

Available In: Client and Server Applications

The TRequestHandlerClass type is used to represent a TRequestHandler class type, and allows TRequestHandlerClass variables to store references to TRequestHandler classes and descendants.

11.114 TScrollBars Type

Unit: WebCtrls

```
TScrollBars = (sbNone,sbVertical,sbHorizontal,sbBoth)
```

Available In: Client and Server Applications

The TScrollBars enumerated type is used with various scrollable controls to specify how scrollbars should be displayed when their content overflows the client area of the control.

Element	Description
sbBoth	Specifies that both a horizontal and vertical scrollbar should be displayed, if necessary.
sbHorizontal	Specifies that a horizontal scrollbar should be displayed, if necessary.
sbNone	Specifies that no scrollbars should be displayed.
sbVertical	Specifies that a vertical scrollbar should be displayed, if necessary.

11.115 TScrollSupport Type

Unit: WebCtrls

```
TScrollSupport = (ssNone,ssVertical,ssHorizontal,ssBoth)
```

Available In: Client and Server Applications

The TScrollSupport enumerated type is used with various scrollable controls to specify the directions in which the controls can be scrolled when their content overflows the client area of the control.

> **Note**
> This property only applies to scrolling via touch and mouse wheel movements, and does not apply to programmatic scrolling or scrolling via the scroll bars themselves.

Element	Description
ssBoth	Specifies that scrolling is allowed in both the horizontal and vertical directions.
ssHorizontal	Specifies that scrolling is allowed in the horizontal direction only.
ssNone	Specifies that scrolling is not allowed in either the horizontal or vertical direction.
ssVertical	Specifies that scrolling is allowed in the vertical direction only.

11.116 TSelectionState Type

Unit: WebBtns

```
TSelectionState = (ssIndeterminate,ssUnselected,ssSelected)
```

Available In: Client and Server Applications

The TSelectionState enumerated type is used with the TCheckBox and TRadioButton classes to specify the selection state of the control.

Element	Description
ssIndeterminate	Specifies that no selection has been made. This is the default value.
ssSelected	Specifies that the control is selected.
ssUnselected	Specifies that the control is not selected.

11.117 TServerRequestErrorEvent Type

Unit: WebHTTP

```
TServerRequestErrorEvent = procedure (Request: TServerRequest;
      const ErrorMsg: String) of object
```

Available In: Client and Server Applications

The TServerRequestErrorEvent event type is used by the TServerRequest OnError event to indicate when a server request could not be completed due to a network error.

The Request parameter indicates the server request that triggered the event. For client applications, the ErrorMsg always indicates the error message determined by the ERR_HTTP_REQUEST_NETWORK translation set in the TFormatSettings Translations property. For server applications, the ErrorMsg is the specific error mesage for the network error.

11.118 TServerRequestEvent Type

Unit: WebHTTP

```
TServerRequestEvent = procedure (Request: TServerRequest) of
      object
```

Available In: Client and Server Applications

The TServerRequestEvent event type is used by the TServerRequest OnComplete event to indicate when a server request is complete.

The Request parameter indicates the server request that triggered the event, and the StatusCode property of the server request can be examined to determine if the request completed successfully.

11.119 TServerRequestMethod Type

Unit: WebHTTP

```
TServerRequestMethod = (rmGet,rmPost,rmHead,rmPut,rmDelete,
      rmPatch)
```

Available In: Client and Server Applications

The TServerRequestMethod enumerated type is used with the TServerRequest component to specify the HTTP method for a web server request.

Element	Description
rmDelete	Specifies that the request is an HTTP DELETE request.
rmGet	Specifies that the request is an HTTP GET request.
rmHead	Specifies that the request is an HTTP HEAD request.
rmPatch	Specifies that the request is an HTTP PATCH request.
rmPost	Specifies that the request is an HTTP POST request.
rmPut	Specifies that the request is an HTTP PUT request.

11.120 TServerRequestProgressEvent Type

Unit: WebHTTP

```
TServerRequestProgressEvent = procedure (Current: Integer;
      Total: Integer) of object
```

Available In: Client and Server Applications

11.121 TServerRequestResponseType Type

Unit: WebHTTP

```
TServerRequestResponseType = (srText,srStream)
```

Available In: Client and Server Applications

The TServerRequestResponseType enumerated type is used by the TServerRequest class to indicate how the response of a web server request should be handled.

Element	Description
srStream	Specifies that the response should be stored in a stream and made accessible via the TServerRequest ResponseStream property.
srText	Specifies that the response should be stored as text and made accessible via the TServerRequest ResponseContent property.

11.122 TServerRequestURL Type

Unit: WebHTTP

```
TServerRequestURL = type String
```

Available In: Client and Server Applications

The TServerRequestURL type is used to represent the URL of a TServerRequest class instance. This type is type-equivalent to a String type, but is used to distinguish the URL of a server request instance when used with special design-time property editors.

11.123 TServerSessionAuthenticateEvent Type

Unit: WebSession

```
TServerSessionAuthenticateEvent = function (Session:
      TServerSession): Boolean of object
```

The TServerSessionAuthenticateEvent event type is used by the TServerSession OnAuthenticate event to allow the developer to provide custom authentication credentials and/or specify if authentication should proceed.

The Session parameter indicates the server session that triggered the event and the Result value determines if the authentication should proceed. If Result value is set to False, then authentication will fail with an exception indicating that authentication was cancelled.

11.124 TServerSessionEvent Type

Unit: WebSession

```
TServerSessionEvent = procedure (Session: TServerSession) of
      object
```

Available In: Client and Server Applications

11.125 TServerSessionProgressEvent Type

Unit: WebSession

```
TServerSessionProgressEvent = procedure (Current: Integer;
      Total: Integer) of object
```

Available In: Client and Server Applications

11.126 TSizerOrientation Type

Unit: WebSizer

```
TSizerOrientation = (soVertical,soHorizontal)
```

Available In: Client and Server Applications

The TSizerOrientation enumerated type is used with the TSizer control to specify in which direction a sizer control be oriented, which determines the direction in which the associated control will be resized as the sizer control is moved.

Element	Description
soHorizontal	The sizer control will size a control in a horizontal direction.
soVertical	The sizer control will size a control in a vertical direction.

11.127 TSlideEvent Type

Unit: WebSlide

```
TSlideEvent = procedure (Sender: TObject; SlideIndex: Integer;
      const SlideImageURL: String) of object
```

Available In: Client and Server Applications

The TSlideEvent type is used by the OnLoadSlide and OnRenderSlide events for the TSlideShow control to intercept the loading or rendering of slide images.

11.128 TSortDirection Type

Unit: WebData

```
TSortDirection = (sdNone,sdAscending,sdDescending)
```

Available In: Client and Server Applications

The TSortDirection enumerated type is used with the TDataColumn class to specify how a column should be sorted in a dataset.

Element	Description
sdAscending	Specifies that the column should be sorted in ascending order.
sdDescending	Specifies that the column should be sorted in descending order.
sdNone	Specifies that the column should not be part of the active sort.

11.129 TStorageChangeEvent Type

Unit: WebComps

```
TStorageChangeEvent = procedure (Sender: TObject; const Key:
      String; const NewValue: String; const OldValue: String; const
      URL: String) of object
```

Available In: Client and Server Applications

The TStorageChangeEvent event type is used by the TPersistentStorage OnChange event to indicate when the persistent local storage (but not the session-only local storage) is changed by another session in the host web browser.

The Key parameter indicates the key of the item that was updated. This parameter will be blank if the contents of the persistent local storage were removed using the ClearAll method.

The NewValue parameter indicates the value of the item that was updated. This parameter will be blank if the contents of the persistent local storage were removed using the ClearAll method, or if the item was removed using the Clear method.

const Key: String;
const NewValue: String; const OldValue: String;
const URL: String

The Request parameter indicates the server request that triggered the event, and the StatusCode property of the server request can be examined to determine if the request completed successfully.

11.130 TStreamOrigin Type

Unit: WebSrvr

```
TStreamOrigin = (orBeginning,orCurrent,orEnd)
```

Available In: Client and Server Applications

The TStreamOrigin enumerated type is used with the TStream Seek method to specify the origin for the seek operation.

Element	Description
orBeginning	Specifies that the stream position should be moved relative to the beginning of the stream.
orCurrent	Specifies that the stream position should be moved relative to the current position in the stream.
orEnd	Specifies that the stream position should be moved relative to the end of the stream.

11.131 TStringsArray Type

Unit: WebCore

```
TStringsArray = array of String
```

Available In: Client and Server Applications

The TStringArray type is used in classes like the TStringList class to represent an array of String values.

11.132 TTextAlignment Type

Unit: WebUI

```
TTextAlignment = (taLeftJustify,taCenter,taRightJustify)
```

Available In: Client and Server Applications

The TTextAlignment enumerated type is used to specify the horizontal alignment of text on a TCanvasElement instance.

Element	Description
taCenter	Specifies that the text will be centered.
taLeftJustify	Specifies that the text will be left-justified.
taRightJustify	Specifies that the text will be right-justified.

11.133 TTextBaseLine Type

Unit: WebUI

```
TTextBaseLine = (blAlphabetic,blTop,blMiddle,blBottom,blHanging,
      blIdeographic)
```

Available In: Client and Server Applications

The TTextBaseLine enumerated type is used to specify the vertical alignment of text on a TCanvasElement instance.

Element	Description
blAlphabetic	Specifies that the baseline is the normal alphabetic baseline. This is the default vertical alignment.
blBottom	Specifies that the baseline is the bottom of the bounding box that encompasses the text.
blHanging	Specifies that the baseline is the hanging baseline.
blIdeographic	Specifies that the baseline is the ideographic baseline.
blMiddle	Specifies that the baseline is the middle of the em square.
blTop	Specifies that the baseline is the top of the em square.

11.134 TTextInputType Type

Unit: WebUI

```
TTextInputType = (tiNone,tiEmail,tiNumber,tiURL)
```

Available In: Client and Server Applications

The TTextInputType enumerated type is used with the TTextInputElement class to specify how the text should be input. This information is used with touch interfaces to determine the type of soft keyboard to display when inputting text into the element.

Element	Description
tiEmail	Specifies that the element will contain an email address.
tiNone	Specifies that the element will contain regular text.
tiNumber	Specifies that the element will contain a number.
tiURL	Specifies that the element will contain a URL.

11.135 TTouchEvent Type

Unit: WebCtrls

```
TTouchEvent = procedure (Sender: TObject; ShiftKey, CtrlKey,
      AltKey: Boolean; X,Y: Integer) of object
```

Available In: Client and Server Applications

The TTouchEvent type is a common event type that is used by controls to provide notification that a touch event is occurring via a touch interface.

The Sender parameter represents the class instance that triggered the event. The ShiftKey, CtrlKey, AltKey parameters represent whether the Shift, Control, and/or Alt keys were also pressed. The X and Y parameters indicate the horizontal and vertical position of the touch, in pixels, relative to the bounds of the control that triggered the event.

11.136 TTouchScrollEvent Type

Unit: WebCtrls

```
TTouchScrollEvent = procedure (Sender: TObject; X,Y: Integer) of
      object
```

Available In: Client and Server Applications

The TTouchScrollEvent type is a common event type that is used by controls to provide notification that a control is being scrolled using a touch movement in any direction.

The Sender parameter represents the class instance that triggered the event. The X parameter indicates the horizontal movement, in pixels, while the Y parameter indicates the vertical movement, in pixels.

11.137 TWebElementEvent Type

Unit: WebUI

```
TWebElementEvent = procedure (AElement: TWebElement) of object
```

Available In: Client and Server Applications

The TWebElementEvent type is used with the TWebElement OnLoad, OnUnload, and OnError events.

11.138 TWebServerRequestErrorEvent Type

Unit: WebSrvr

```
TWebServerRequestErrorEvent = function (const Message: String):
      Boolean of object
```

Available In: Client and Server Applications

11.139 TZoomControlStyle Type

Unit: WebMaps

```
TZoomControlStyle = (zcDefault,zcLarge,zcSmall)
```

Available In: Client and Server Applications

The TZoomControlStyle enumerated type is used with the TZoomControlOptions class to specify the size of the zoom control.

Element	Description
zcDefault	The default size (large).
zcLarge	A large zoom control.
zcSmall	A small zoom control.