



Embarcadero® RAD Studio XE

VCL for the Web (IntraWeb)

All IDE packages loaded

embarcadero

Copyright© 2010 Embarcadero Technologies, Inc. All Rights Reserved.

Delphi XE IntraWeb XI Development

Delphi XE, IntraWeb XI (VCL for the Web)
1th EDITION for Bob Swart <Bob@eBob42.com>

42
eBob42.com

Bob Swart (aka Dr.Bob)
Bob Swart Training & Consultancy (eBob42)
<http://www.eBob42.com>



Table of Contents

1. VCL for the Web / IntraWeb XI	1
IntraWeb XI 11.x and Delphi XE	1
IDE Support.....	1
Discontinued Features	1
IntraWeb XI Features	2
Uninstall Previous Version	3
Installing IntraWeb XI	5
IntraWeb License Keys.....	11
Summary	12
2. IntraWeb XI Applications	13
IWDemo Project Source	14
GUI Mode	15
Service Mode	16
Server Controller.....	17
TIWServerController Properties	19
AllowMultipleSessionsPerUser	20
AppName	20
AuthBeforeNewSession	20
Auther	20
BoundIP	22
CacheDir	22
CacheExpiry	22
CharSet.....	22
ComInitialization	22
Compression.....	22
ContentFiles	23
DebugHTML	24
Description	24
DisplayName	24
EnableImageToolbar	24
ExceptionDisplayMode	24
FilesDir	24
HistoryEnabled.....	24
HTMLHeaders.....	24
InternalFilesDir	24
InternalFilesURL.....	25
JavascriptDebug.....	25
Log	25
MasterTemplate	25
PageTransitions.....	25
Port	25
RedirectMsgDelay.....	25
ServerResizeTimeout	26
SessionTimeout.....	26
ShowLoadingAnimation	26
SSLOptions.....	26
StyleSheet	26
TemplateDir.....	26
TimeoutResponse	26
TIWServerController Events	27
OnAfterDispatch.....	27
OnAfterRender	27

OnBackButton.....	27
OnBeforeDispatch.....	28
OnBeforeRender.....	28
OnCloseSession.....	28
OnException.....	28
OnGetSessionID.....	29
OnNewSession.....	29
IntraWeb TIWApplication Properties.....	29
ActiveForm.....	30
ActiveFormCount.....	30
AppID.....	30
ApplicationURL.....	30
Browser.....	30
Data.....	30
FormAction.....	30
FormCount.....	30
Forms.....	30
IP.....	30
IsCallback.....	30
LastAccess.....	30
RedirectURL.....	30
ReferringURL.....	31
Request.....	31
Response.....	31
RunParams.....	31
SecureMode.....	31
SessionTimeout.....	31
Terminated.....	31
TerminateMessage.....	31
TerminateURL.....	31
TrackID.....	31
UserCacheDir.....	32
TIWApplication Methods.....	32
GoToURL.....	32
MarkAccess.....	32
SendFile.....	32
SendStream.....	32
ShowMessage.....	32
Terminate.....	33
TerminateAndRedirect.....	33
TIWAppForm.....	33
IntraWeb TIWPageForm.....	33
ActiveControl.....	35
Background.....	35
ExtraHeader.....	35
HandleTabs.....	35
HiddenFields.....	35
JavaScript.....	35
LayoutMgr.....	35
LinkColor.....	35
ShowHint.....	35
StyleSheet.....	35
SupportedBrowsers.....	35
TextColor.....	35
Title.....	35
VLinkColor.....	35

TIWPageForm Events.....	36
OnAfterRender	36
OnCreate.....	36
OnDefaultAction.....	36
OnDestroy	36
OnRender	36
Designing IntraWeb Page Form	37
IW Standard Controls	37
TIWApplet	37
TIWButton.....	38
TIWCheckBox	39
TIWComboBox	40
TIWEdit.....	41
TIWFile	41
TIWFlash.....	41
TIWHRule.....	42
TIWImage	42
TIWImageFile	42
TIWImageButton.....	42
TIWList	42
TIWLabel.....	42
TIWListbox	43
TIWLink	43
TIWMemo.....	44
TIWMenu.....	44
TIWProgressBar	44
TIWRadioGroup.....	45
TIWRectangle	45
TIWRegion.....	45
TIWText	45
TIWTimer	45
TIWGrid	46
TIWTreeView	47
TIWTreeViewItem	47
TIWURL.....	47
TIWURLWindow.....	47
TIWMPeg.....	47
TIWQuickTime	47
TIWCalendar.....	47
Multiple IntraWeb Application Forms	48
Final Release	48
Passing Information Around	49
Getting Back.....	49
State Management	50
UserSession.....	50
Extending User Session.....	51
Using User Session	51
Summary	52
3. IntraWeb and Databases.....	53
Data Module	53
TSQLConnection.....	54
TSQLDataSet	55
IW Data Controls.....	56
TIWDBCheckBox	56
TIWDBComboBox	56
TIWDBEdit.....	56

TIWDBGrid	56
TIWDBImage	56
TIWDBLabel.....	56
TIWDBListbox	57
TIWDBLookupListbox	57
TIWDBLookupCombobox	57
TIWDBFile	57
TIWDBMemo.....	57
TIWDBNavigator	57
TIWDBText	57
TIWDBRadioGroup.....	57
Continued Demo	58
TIWDBGrid Usage.....	62
Sharing VCL data modules with VCL for the Web	67
Pool Data Connections	68
DatamoduleUnit	69
ServerController.....	69
Using Data Pooling	71
Custom Data Pooling	72
Summary	72
4. IntraWeb and AJAX.....	73
AJAX = Asynchronous	73
OnAsync.....	73
OnAsync Events	75
EventParams	76
Working with EventParams.....	78
OnAsync and Visible	79
OnAsync and Disable	79
Summary	80
5. IntraWeb and iPhone / iPad.....	81
TMS IntraWeb iPhone Controls Pack	81
TTIWiPhoneButton.....	83
TTIWiPhoneEmailLabel	83
TTIWiPhoneFooter	83
TTIWiPhoneGeolocation	84
TTIWiPhoneHeader	84
TTIWiPhoneList	84
TTIWiPhoneLocationLabel	84
TTIWiPhoneMenu	84
TTIWiPhoneOnOffButton	84
TTIWiPhonePageFlip	85
TTIWiPhonePhoneLabel	85
TTIWiPhoneRegion	85
TTIWiPhoneScrollRegion.....	85
TTIWiPhoneSMSLabel	85
TTIWiPhoneStyle.....	85
TTIWiPhoneTrackbar	85
TMS iPhone Controls Demo.....	86
Registered Users	95
Summary	96
6. IntraWeb Custom Components.....	97
IntraWeb Controls	97
Custom Components.....	99
Packages.....	100
TIEuroComboBox	102

TIWRequiredEdit	104
Installation and Usage	105
Summary	105
7. IntraWeb Testing Framework.....	107
Sample Application	107
Manual Test.....	108
VCL for the Web Test Project	109
Sharing Main Form	110
Writing Test Code.....	110
More Tests.....	117
ITestSuite	118
Summary	119
8. IntraWeb Deployment.....	121
Project Targets.....	121
StandAlone Application	121
Service Application	122
ISAPI Extension	122
Multiple Project Targets	122
Windows Server 2003	123
Enabling ISAPI / CGI	123
Virtual Directory.....	124
Deployment on Windows Server 2008 and IIS7	127
IntraWeb Deployment.....	132
Files, Templates and Cache	132
Database Drivers.....	132
DBX4 Drivers	132
DBX Trace / Pool Connections.....	133
Core Lab DBX4 Drivers.....	133
Summary	134

The information in this courseware manual is © 2001-2011 by drs. Robert E. (Bob) Swart of Bob Swart Training & Consultancy. All Rights Reserved.

The information in this courseware manual is presented to the best of my knowledge at the time of writing. However, in case of errors or omissions, I welcome your feedback or comments (by e-mail) as Bob Swart Training & Consultancy cannot be held responsible for any damage that results from using the information in this manual or the example source code snippets. Thanks in advance for your understanding.

1. VCL for the Web / IntraWeb XI

IntraWeb is a third-party product which has been included with Delphi since version 7, has been renamed to “VCL for the Web” since Delphi 2007, and offers a RAD way to build Win32 web applications, reusing data access components and data modules when needed. Including support for AJAX and custom controls, and third-party controls from TMS Software to build even web applications for the iPhone and iPad for example.

In this first section, I will cover how to correctly install IntraWeb XE on your machine and integrate it with your copy of Delphi. But we also examine the different editions and features of IntraWeb XI.

IntraWeb XI 11.x and Delphi XE

Since Delphi 7, IntraWeb ships as part of Delphi in the box. Delphi 7 came with IntraWeb 5.0 for example (and also included a free upgrade to IntraWeb 5.1 – but after that you had to purchase a license for further updates). Delphi XE comes with IntraWeb XI version 11.0.0 in the box. With Delphi XE Professional you get IntraWeb XI Personal, and with Delphi XE Enterprise you get IntraWeb XI Standard. Note that these are not the full “ultimate” versions of IntraWeb.

Also, even if you want to keep using the bundled Personal or Standard version, it is strongly recommended to register yourself at the AtoZedSoftware website to get your free key which will entitle you to free updates of your edition of IntraWeb XI –not the major upgrades – and the bundled features, but not the full ultimate functionality.

See <http://www.atozed.com/IntraWeb/FeatureMatrix.aspx> for an overview of the features found in IntraWeb XI, which are also briefly summarised here.

IDE Support

IntraWeb XI is supported by Delphi 7, 2006, 2007, 2009, 2010 and Delphi XE:

	Evaluation	Personal	Standard	Ultimate
Supported Environments	Delphi 7, 2006, 2007, 2009, 2010, XE	Delphi 7, 2006, 2007, 2009, 2010, XE	Delphi 7, 2006, 2007, 2009, 2010, XE	Delphi 7, 2006, 2007, 2009, 2010, XE
Bundled With	available as free download	Delphi XE or RAD Studio XE Professional	Delphi XE or RAD Studio XE Enterprise / Architect	only available as separate purchase

See the above URL for pricing and special offers with regards to the Personal, Standard and Ultimate editions of IntraWeb XI. In this courseware manual, I will use Delphi XE Enterprise in combination with IntraWeb XI Ultimate.

Discontinued Features

For people who worked with previous versions of IntraWeb, it may be important to note the features and functionality which are no longer found in IntraWeb XI.

Page Mode has been deprecated, and replaced by Integrated Page Mode. Unfortunately, while Integrated Page Mode offers new capabilities, the “old” Page Mode options are no longer possible. So, if you have existing WebBroker Page Mode application using IntraWeb 10.x or lower, you may need to maintain them using the older version of IntraWeb (potentially on a different machine if you also want to use IntraWeb XI in the same version of Delphi).

Support for WAP and HTML 3.2 is also removed from IntraWeb XI. As a result, there is now only one main form type: for HTML 4 and higher. If you still have WAP or HTML 3.2 web applications using IntraWeb 10.x or lower, you may need to maintain them using the older version of IntraWeb (potentially on a different machine if you also want to use IntraWeb XI in the same version of Delphi).

To be honest, this feature is probably not used a lot anymore, as most mobile devices support a rich set of HTML, and we'll see in the section on IntraWeb and iPhone / iPad development for example.

As another consequence of this, support for older browser has also been removed from IntraWeb XI. The (minimum version of) supported browsers are as follows (no specific support for Opera exists anymore):

- FireFox 3
- Internet Explorer 7
- Chrome, Safari

The Client Side components, like Client Side DataSets Client Side Charts, and DynGrid have been removed from IntraWeb XI. These features are no longer used, and can be replaced with AJAX functionality.

On a similar note, the feature using partial updates has been deprecated, and should be replaced with full use of AJAX now. AJAX support was introduced in IntraWeb 9.x, and there should be no use of umPartial – if you still use it, then just replace it by OnAsync event handlers instead for much better results.

Support for Apache DSO files is discontinued for two reasons: first of all, Delphi hasn't been offering Apache support in WebBroker Wizards for some time (although it's unofficially still available in the source code units), and Apache can now run ISAPI DLLs so there is no need for special DSO support from IntraWeb.

The session in IntraWeb application used to be tracking using the SessionID in the URL, a hidden field, or using a cookie (the tmURL, tmHidden and tmCookie options). IntraWeb XI will support cookies for session tracking, and this has also lead to the new feature called Integrated Page Mode and URL mapping.

As a consequence of using cookies to track the session, it may no longer be possible to allow a single user to run multiple sessions in separate tabs or windows (since the cookie may be shared by these tabs). For this purpose, IntraWeb XI introduces the property AllowMultipleSessionsPerUser in the ServerController, which will ensure that each browser window or tab can have its own unique session cookie (so you can have multiple user sessions).

IntraWeb XI Features

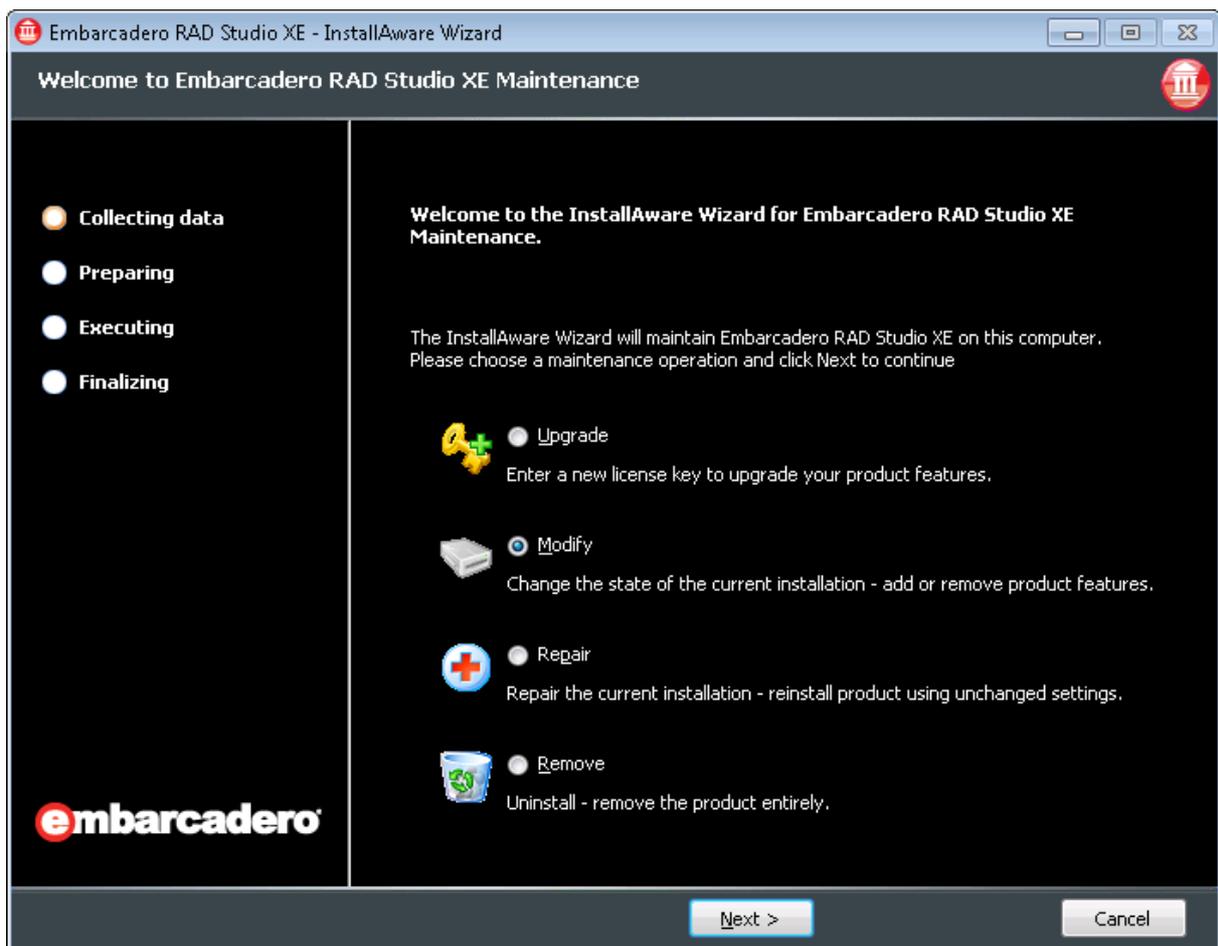
The following table lists the features of IntraWeb and the four different editions.

	Evaluation	Personal	Standard	Ultimate
Standalone Server Deployment	Yes	Yes	Yes	Yes
Service Deployment			Yes	Yes
ISAPI Deployment				Yes
SSL Encryption				Yes
IP Binding				Yes
Priority Support				Yes
Access to (most) Source Code				Yes
Concurrent Session Limit		5		
Application Mode	Yes	Yes	Yes	Yes
XI Authentication	Yes			Yes
Session Inactivity Timeout	Config	20 mins	20 mind	Config

Even if you use the bundled edition, you could decide to purchase IntraWeb XI Ultimate. For this courseware manual, I've used IntraWeb XI Ultimate version 11.0.32.

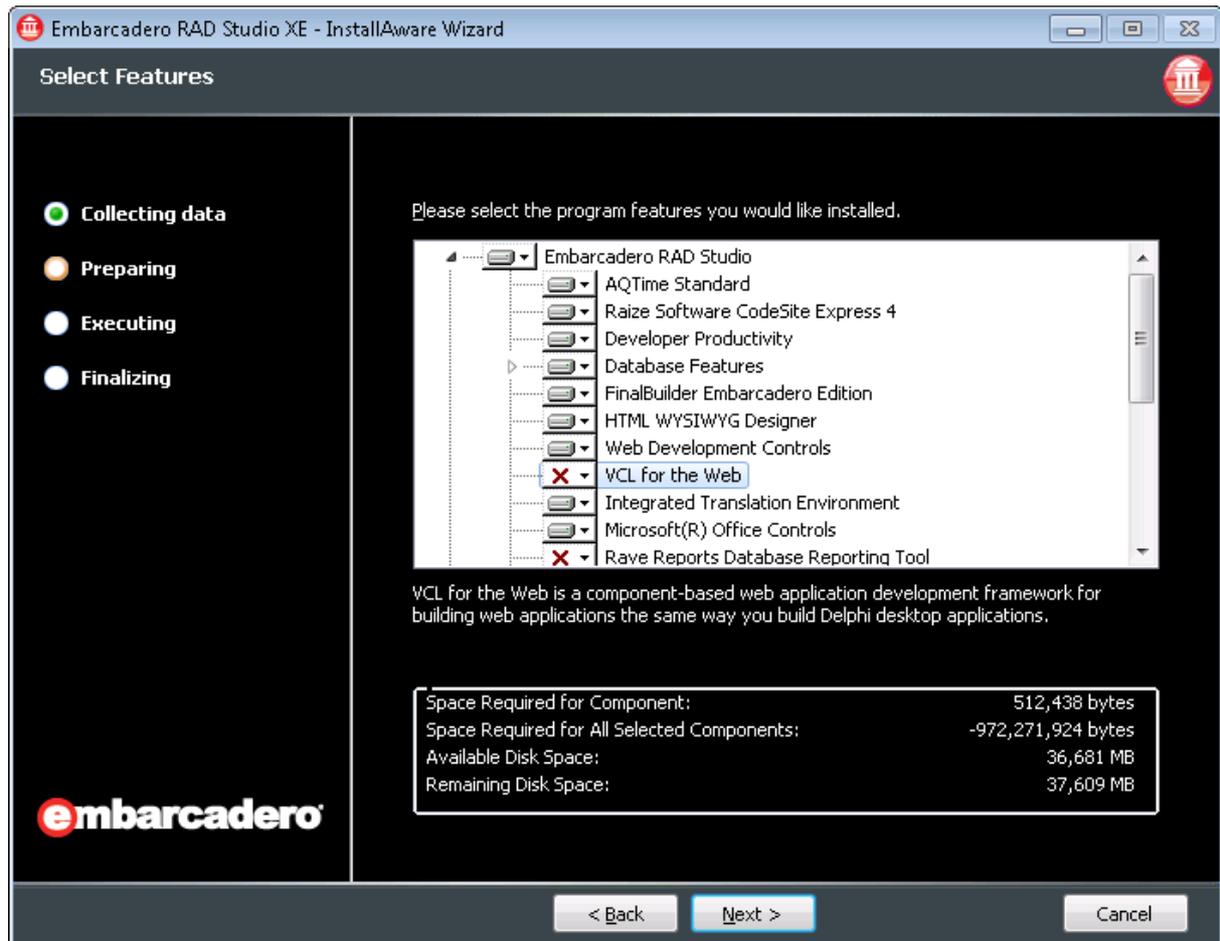
Uninstall Previous Version

Before you install the latest version of IntraWeb, you should make sure the bundled edition is uninstalled from Delphi (or not installed in the first place). If you are uncertain whether or not you included IntraWeb (VCL for the Web) in the installation of Delphi XE, or you installed it and need to remove it, then you can start the "Modify, Repair, Uninstall" item from the Embarcadero RAD Studio XE program group. This will call the C:\ProgramData\{7DE921C9-42C8-4DA3-8A44-043C3349FD1D}\Setup.exe and present you with the RAD Studio XE Maintenance dialog. Here, you can select Modify in order to modify the existing installation:



Click on Next, and after you've selected Delphi XE (and optionally C++Builder XE if your serial number also enabled that) in the second page, you can click on Next again to enter the third page of the wizard where you can select the languages that will be installed on your machine.

The next page will show the features of Delphi XE that are included with your edition (the list of features will differ between Delphi XE Professional and Delphi XE Enterprise). Make sure that the VCL for the Web item is unselected, so it will be removed from your system if it was already installed. If the red cross is already showing, to indicate that it's not installed on your machine at this time, then you can cancel the dialog. Otherwise, you need to press Next again and continue to allow the installer to modify your installation of Delphi XE and remove the bundled version of IntraWeb XI from your disk.



After this process has finished, you should check the Windows\System32 directory to make sure there are no IntraWeb_110_150.bpl and IntraWebDB_110_150.bpl packages left.

Same with the C:\Program Files\Embarcadero\RAD Studio\8.0\bin directory, which should no longer contain the dllIntraWeb_110_150.bpl design time package.

And the C:\Program Files\Embarcadero\RAD Studio\8.0\lib\win32\release directory should no longer contain any IW*.dcu or IW*.res files.

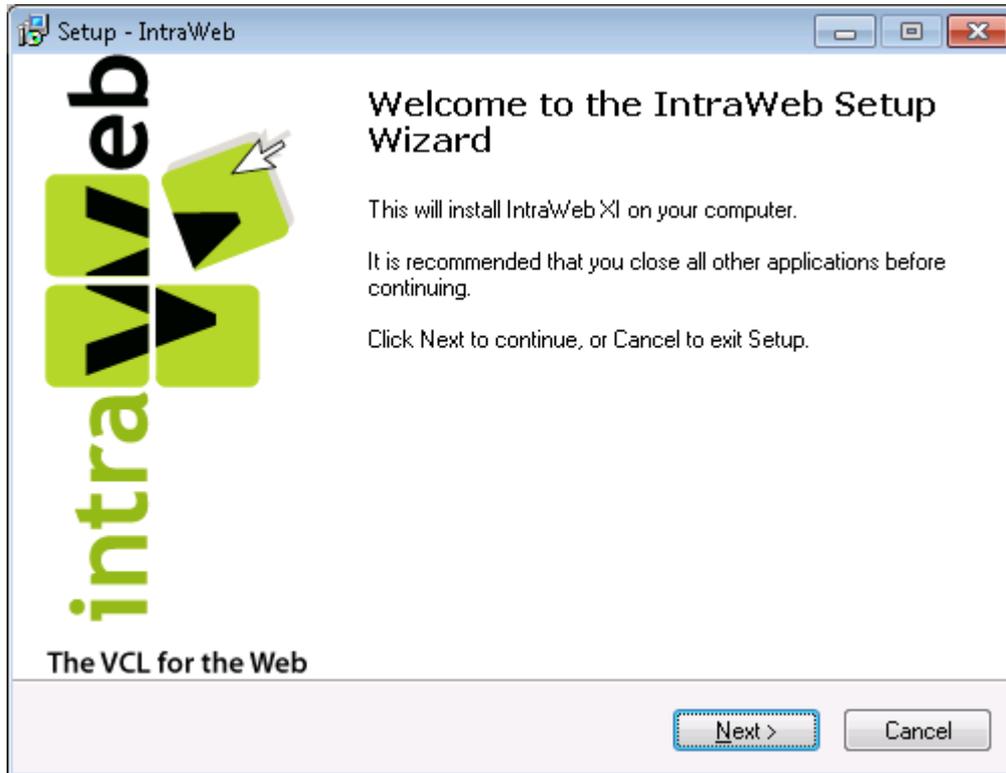
If one or more of these conditions fail, and you still find evidence of the bundled version of IntraWeb XI on your machine, then you could either try to uninstall it again (using the above steps) or remove the files manually from your disk – at your own risk, of course, since uninstalling should remove them.

However, the best way to remove the bundled version of IntraWeb XI from your installation of Delphi XE is simply to install Delphi XE on a clean machine without having selected IntraWeb XI.

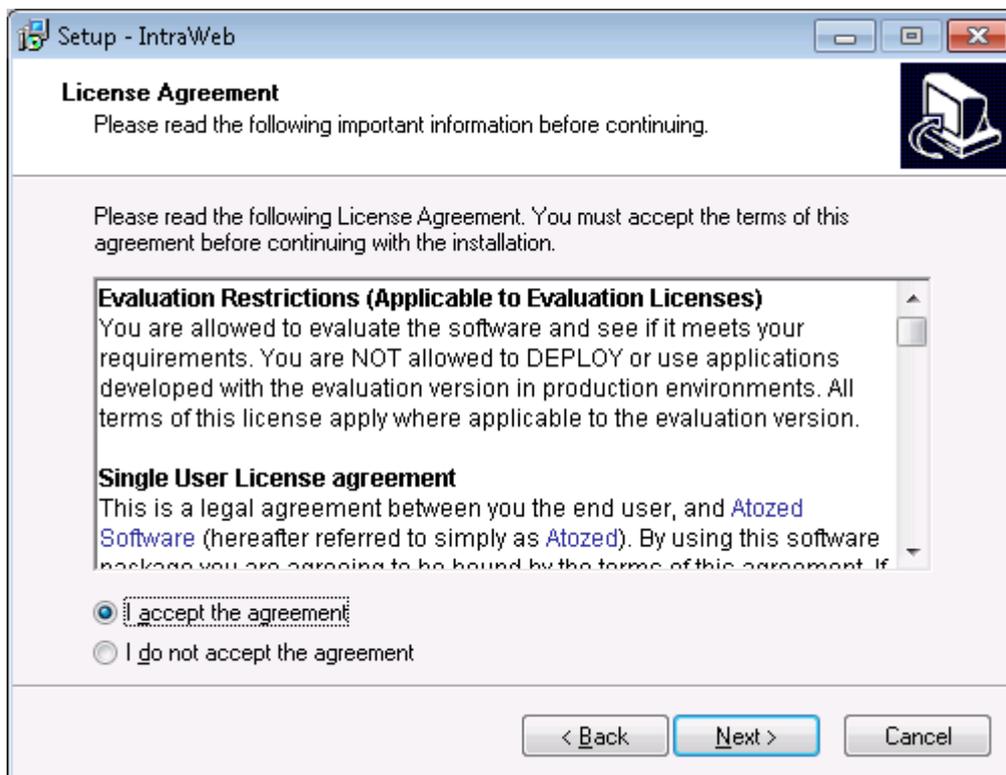
Regardless of how you removed the bundled (or previous) version of IntraWeb XI from your machine, you can then download the latest version of IntraWeb XI from the AtoZedSoftware website at <http://www.atozed.com/IntraWeb/Download/Download.aspx>. At the time of writing, this is IntraWeb XI 11.0.32.

Installing IntraWeb XI

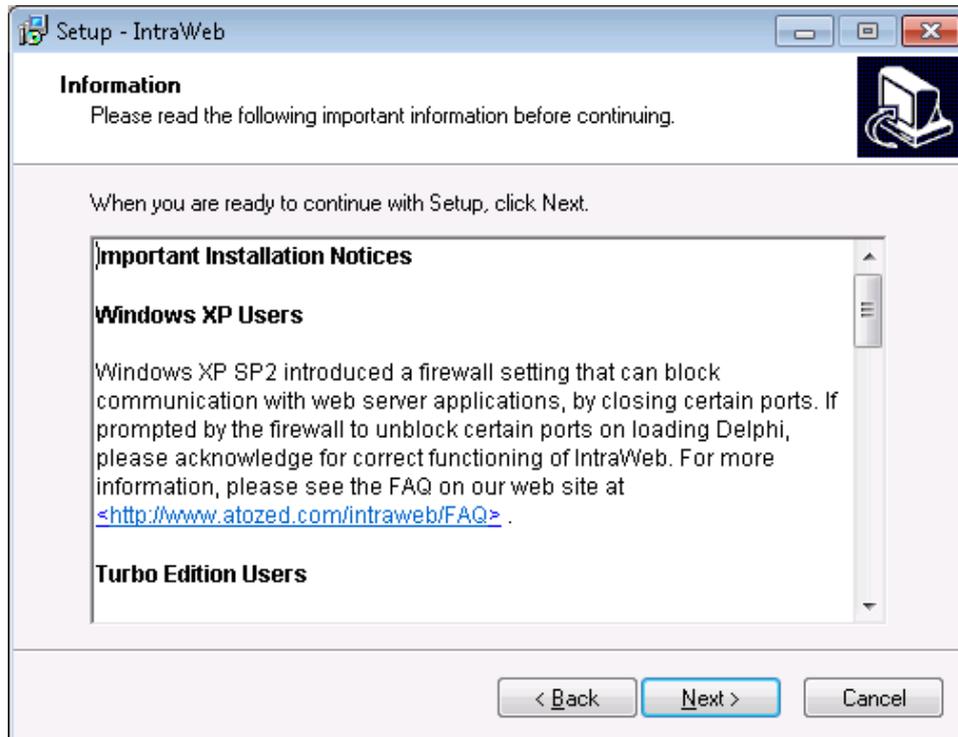
The IntraWeb XI installer recommends to close all other application – at least Delphi must be closed before you install.



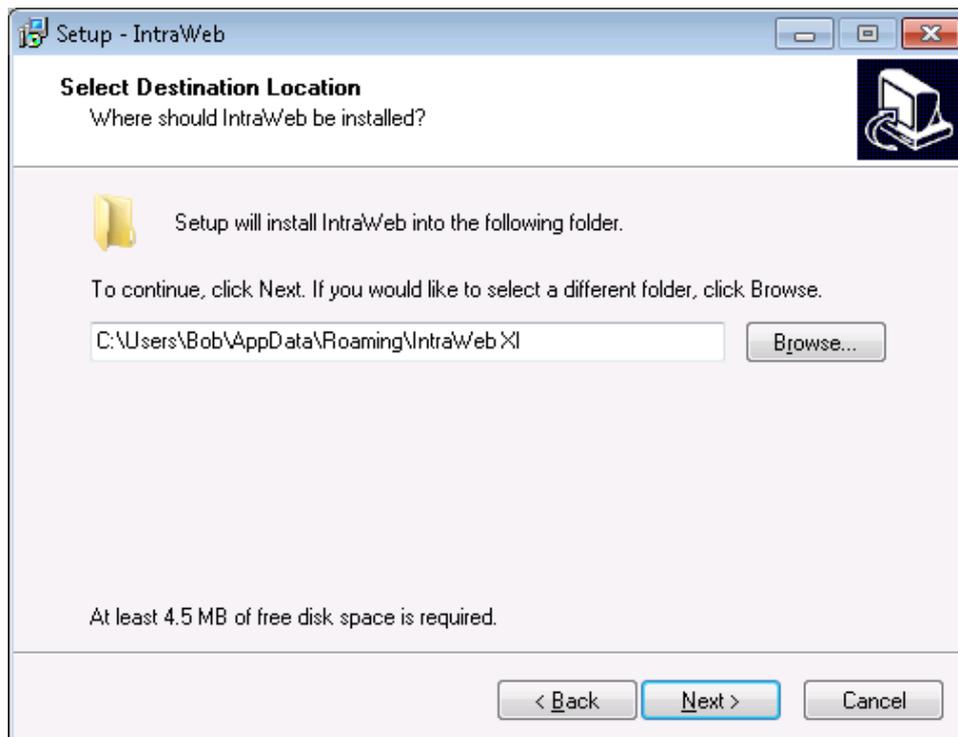
The License Agreement of IntraWeb reminds you that the Personal, Standard and Ultimate licenses are single user licenses only.



The next step gives some important installation notices, which include a note about Windows firewall which may give a warning when you start Delphi or an IntraWeb application written in Delphi.

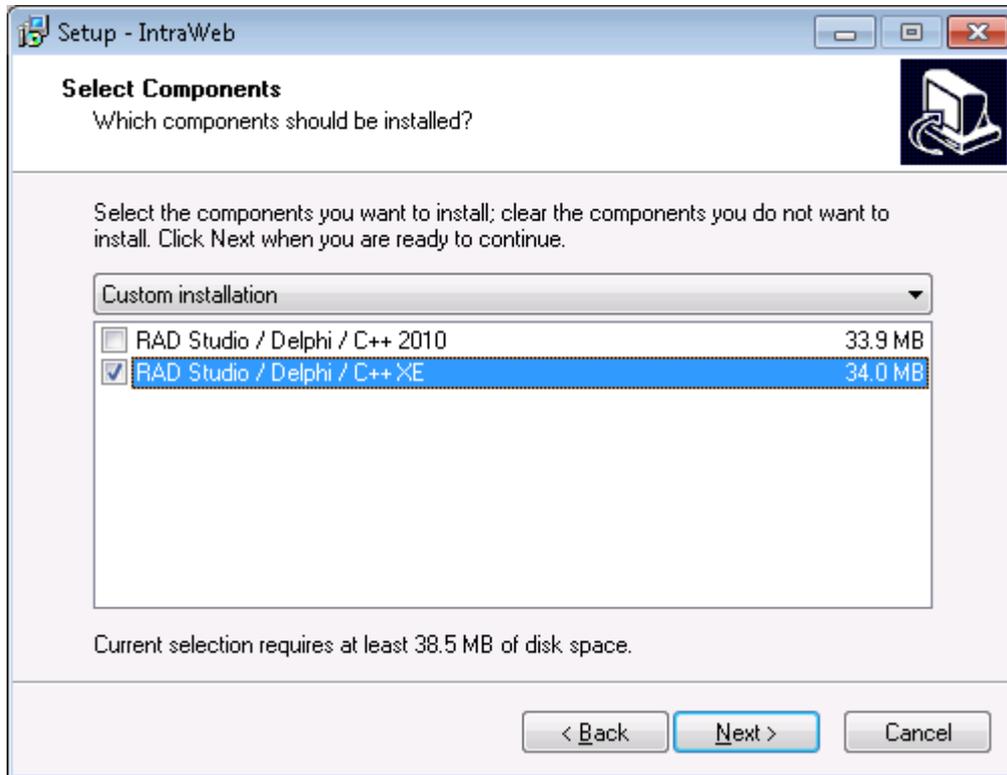


The next page can be used to specify the location of IntraWeb XI. Note that it's no longer installed by default in the Program Files directory, but rather in the C:\Users\CURRENTUSER\AppData\Roaming\IntraWeb XI (where CURRENTUSER is the name of the current user, obviously, or "Bob" in my case).

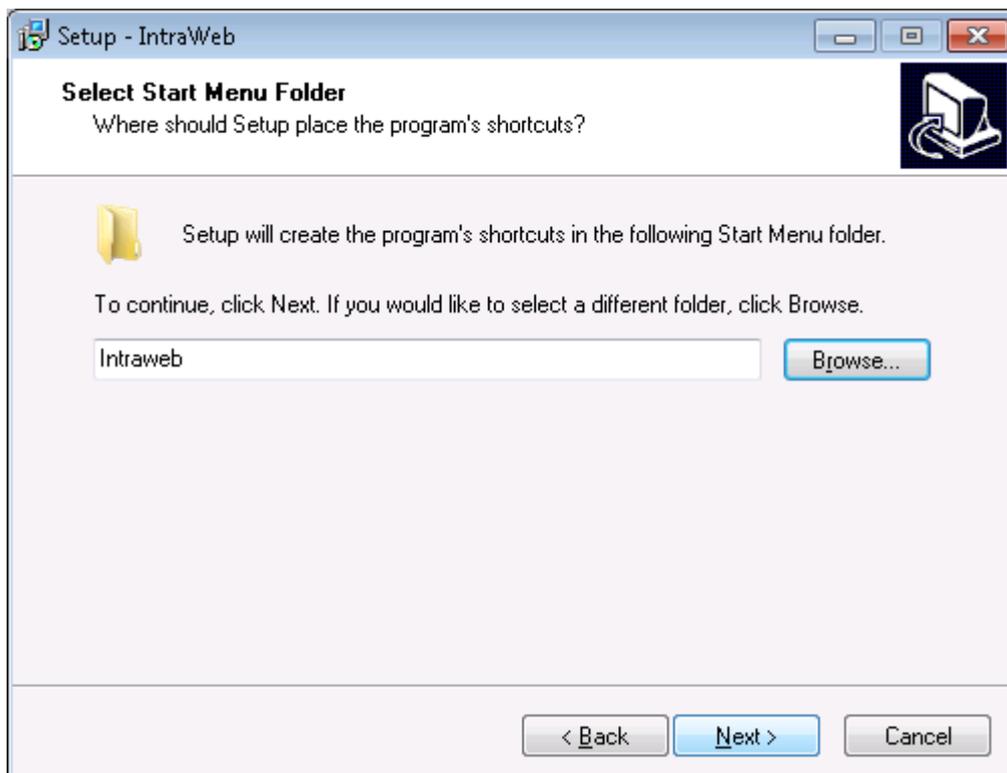


Depending on the versions of Delphi on your machine, the next page will give you the option to select the supported versions of Delphi; Delphi 7, Delphi 2006, Delphi 2007, Delphi 2009, Delphi 2010 or Delphi XE.

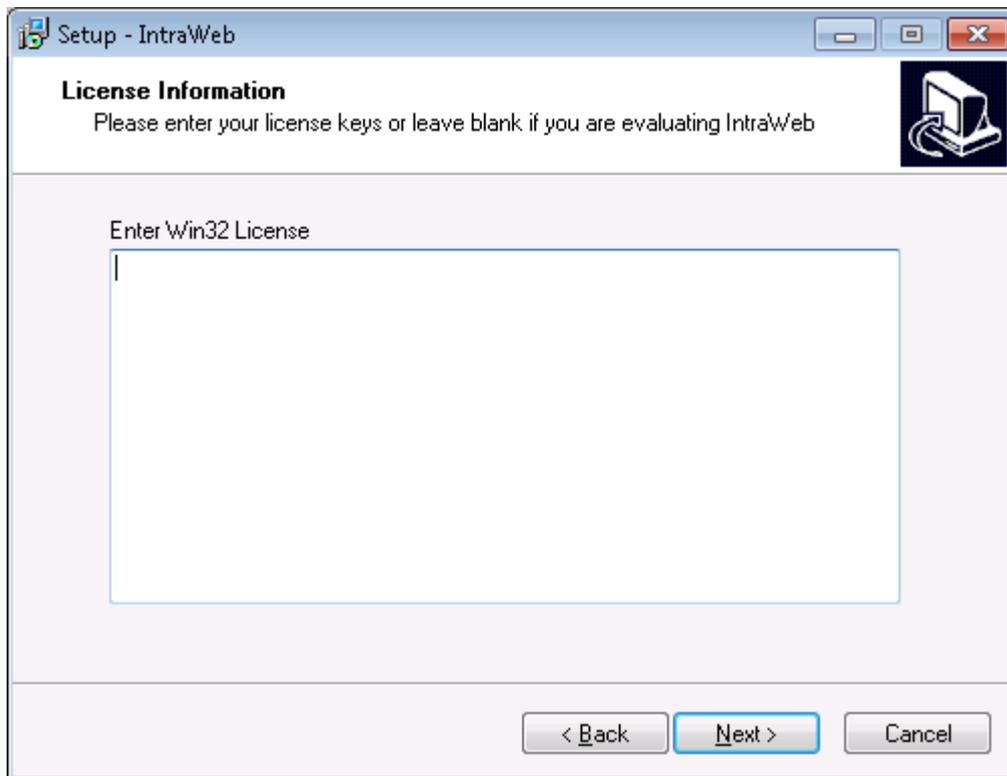
Since I still want to use IntraWeb 10.x with Delphi 2010 (because of some legacy IntraWeb applications), I only select Delphi XE as target to install IntraWeb XI for.



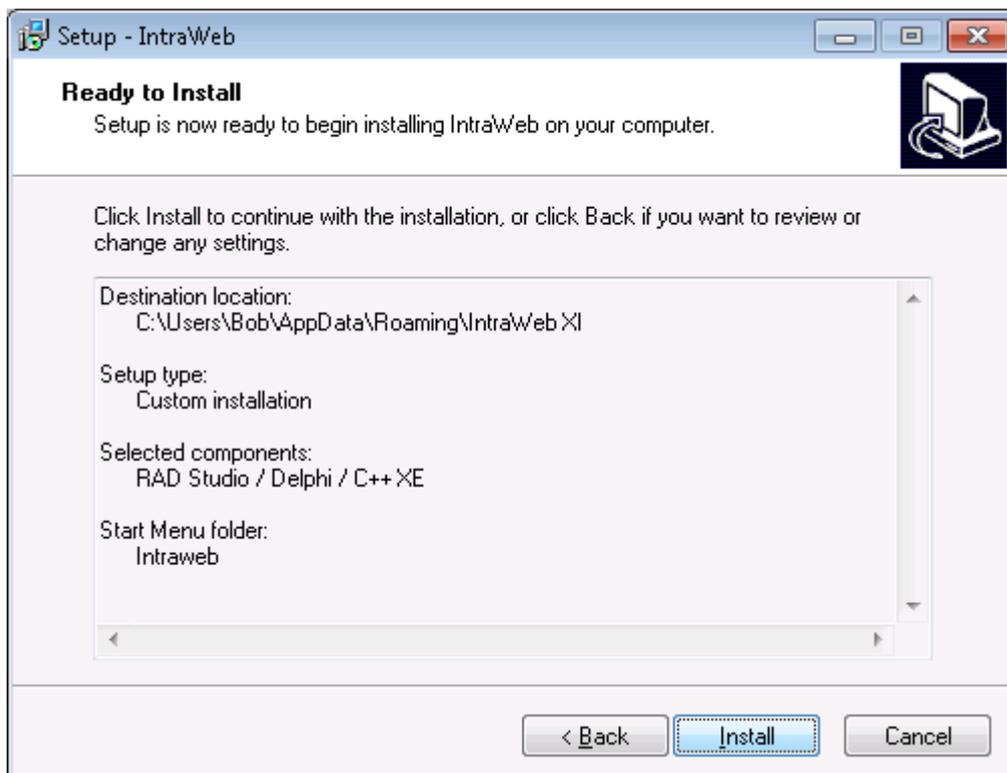
The start menu folder is called IntraWeb by default.



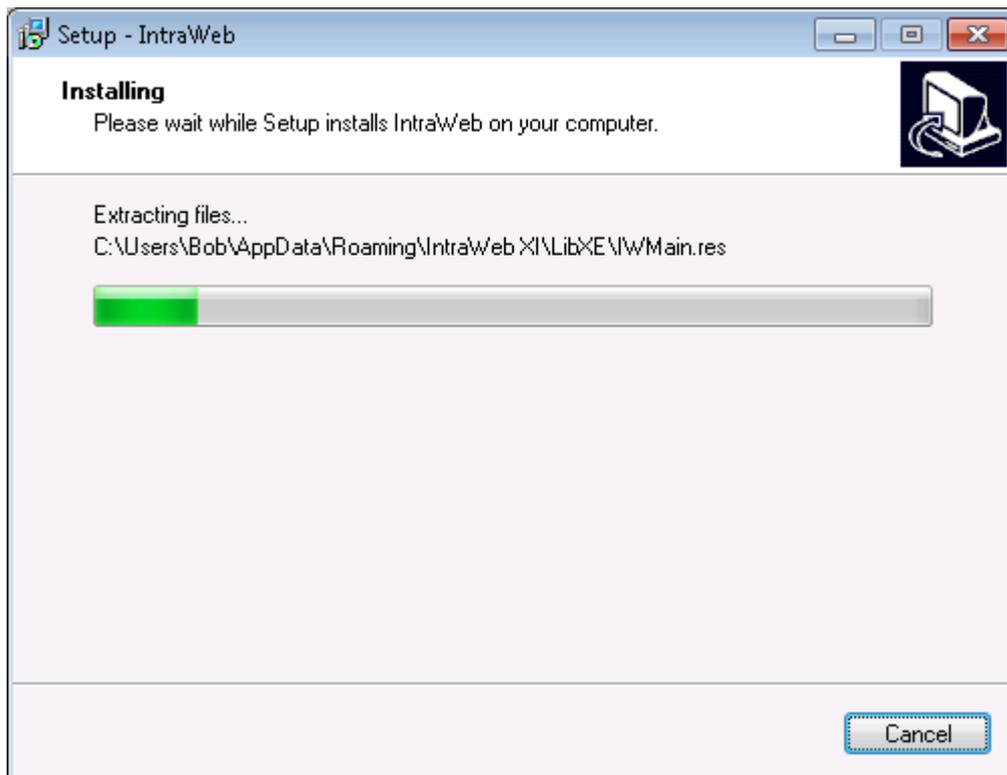
The next page requires your IntraWeb XI key. This key starts with +008 (and not +007 which was used for previous versions of IntraWeb). Even as a bundled user, you must request your key at <http://www.atozed.com/IntraWeb/Download/FreeKeyRequest.aspx> by downloading http://downloads.atozed.com/intraweb/KeyRequest_RadStudioXE1.zip



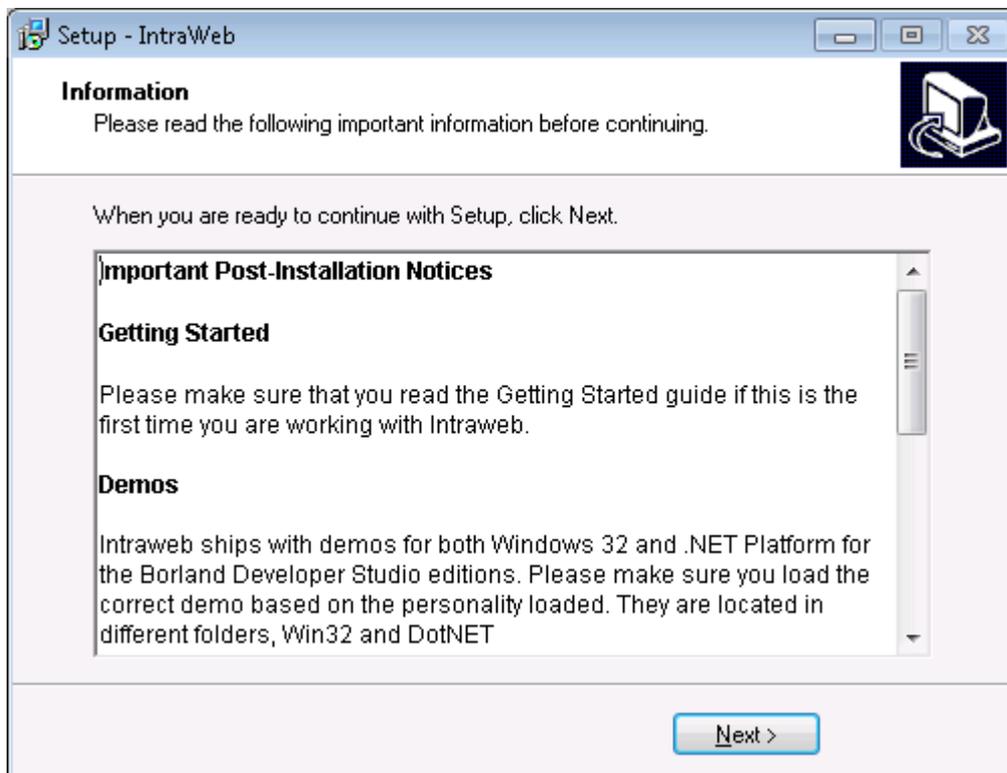
Below is the overview of the settings of IntraWeb XI Ultimate 11.0.23 which was installed on my machine with Delphi XE Enterprise.



Installation will only take a few minutes.



After installation is completed, you will see some post-installation notes. Most of these are outdated. There are only Win32 demos, since .NET is no longer supported for example.



This information should also mention <http://iwxdemos.codeplex.com/> as location for some IntraWeb demos.



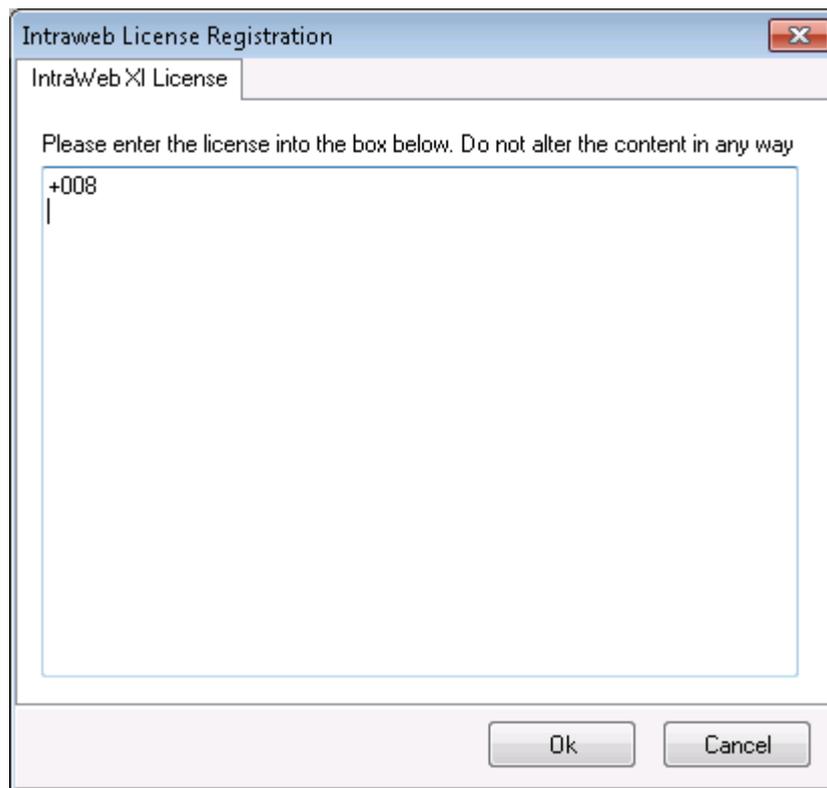
After installing IntraWeb XI, you can check the status and version by starting Delphi XE again, and checking the About Box.



IntraWeb License Keys

After the installation of IntraWeb XI, if you entered your correct license key numbers during installation, you should find a file called IWLICENSEKey.pas in the different library directories of C:\Users\Bob\AppData\Roaming\IntraWeb XI\. For my Delphi XE installation, this means C:\Users\Bob\AppData\Roaming\IntraWeb XI\LibXE.

If you didn't enter the license key during the installation of the full version, or if you started with an evaluation version and want to turn it into a full version (or if you want to re-start an expired subscription), you need to generate the IWLICENSEKey.pas file yourself. This can be done using the LicenseRegistration.exe application that is included with IntraWeb in the C:\Users\Bob\AppData\Roaming\IntraWeb XI\ directory. When you run LicenseRegistration, you only have to paste the Win32 license key:



This application will generate the IWLICENSEKey.pas files for you. It will have the following layout (so you can always reproduce this file yourself if you have your key):

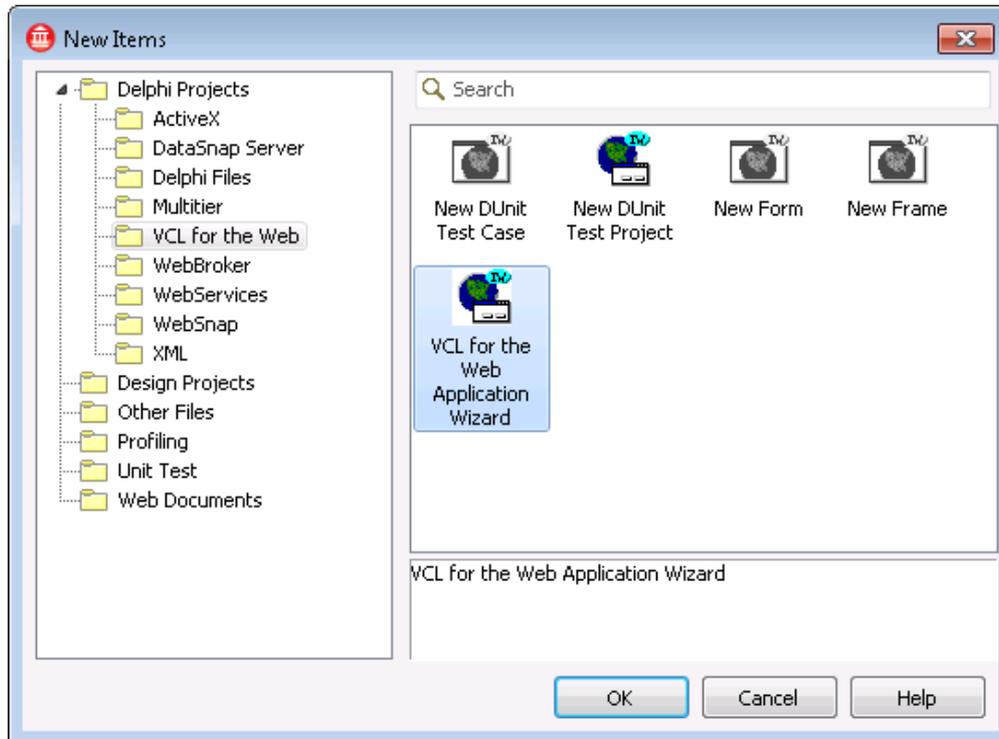
```
unit IWLICENSEKey;  
interface  
  
implementation  
uses  
    IWGlobal;  
  
initialization  
    SetLicenseKey(  
        '+008'+  
        '#####'  
        '#####'  
    );  
end.
```

Summary

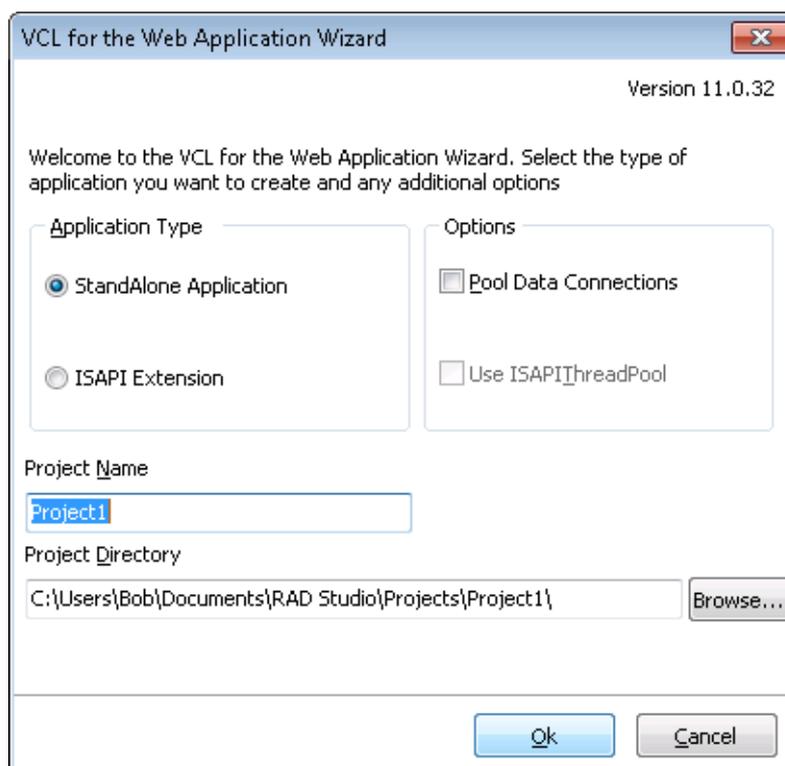
In this section, we have examined the different editions and features of IntraWeb XI. We have covered how to correctly uninstall a previous version of IntraWeb XI, and how to install IntraWeb XE on your machine and integrate it with your copy of Delphi.

2. IntraWeb XI Applications

IntraWeb XI can be used to design WYSIWYG web applications. The best way to demonstrate this, it to start Delphi XE, do *File | New - Other*, go to the VCL for the Web category in the Object Repository, where we can find the two IntraWeb wizards (and three disabled icons):



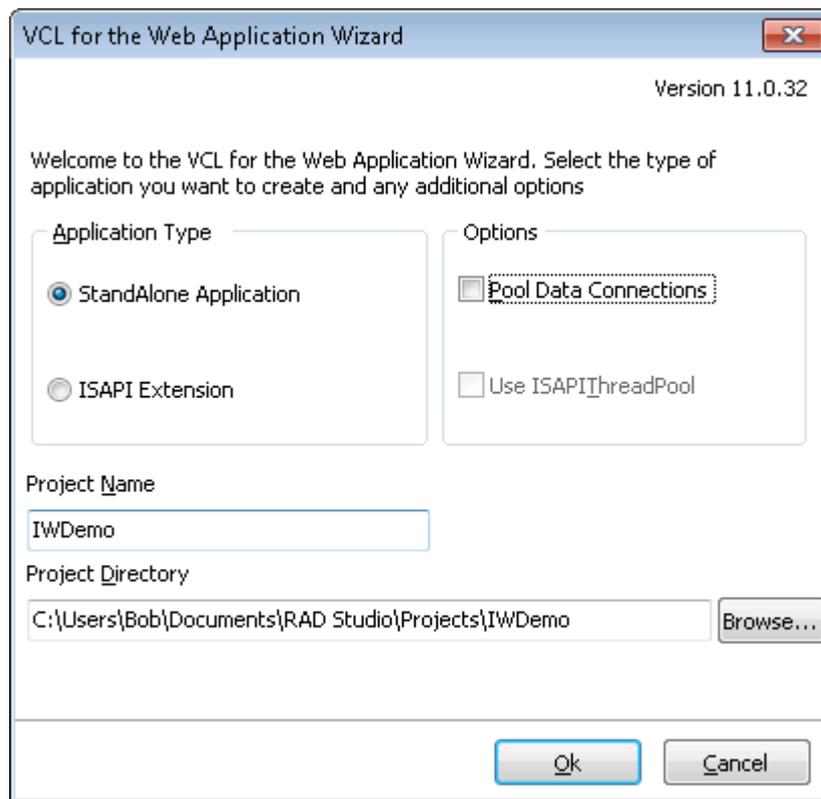
Double-click on the VCL for the Web Application Wizard to start the dialog below.



Looking at the dialog, you may wonder if there are only two possible targets, and no more support for a Windows Services. Looks are deceiving, however, since the standalone application can be turned into a Windows Service (and is in fact a Windows Service by running it with the /GUI command-line option).

The ISAPI Extension option is available, but can only be deployed for developers using the IntraWeb XI Ultimate license. Be aware of that fact when building a web application, as the ISAPI deployment will fail when using the Personal or Standard edition of IntraWeb. The option to use an ISAPI Threadpool is only relevant when producing an ISAPI DLL for IIS, and disabled otherwise. The choice for pooling data connections will be relevant in the next section (about IntraWeb and Databases) and can be left unchecked for now.

Specify IWDemo as Project Name, which will produce a new IntraWeb XI project in the C:\Users\Bob\Documents\RAD Studio\Projects\IWDemo directory:



If you click on OK, then a new IntraWeb standalone application project will be created, as well as three additional units - one for the Server Controller in ServerController.pas, one for the User Session in UserSessionUnit.pas, and one for the IntraWeb Application Form, which can be saved in IWFormMain.pas.

IWDemo Project Source

Let's first examine the generated IWDemo project source code in IWDemo.dpr. Do *Project | View Source* to see the following code:

```
program IWDemo ;

uses
  Forms,
  IWStart,
  UTF8ContentParser,
  IWFormMain in 'IWFormMain.pas' {IWForm1: TIWAppForm},
```

```
ServerController in 'ServerController.pas'  
  {IWServerController: TIWServerControllerBase},  
UserSessionUnit in 'UserSessionUnit.pas' {IWUserSession: TIWUserSessionBase};  
  
{ $R *.res }  
  
begin  
  TIWStart.Execute(True);  
end.
```

There is no more IWMain unit or IWInitService unit required, and also no Application that needs to be initialized and run. Instead, this has been replaced by an Execute call to TIWStart, from the IWStart unit.

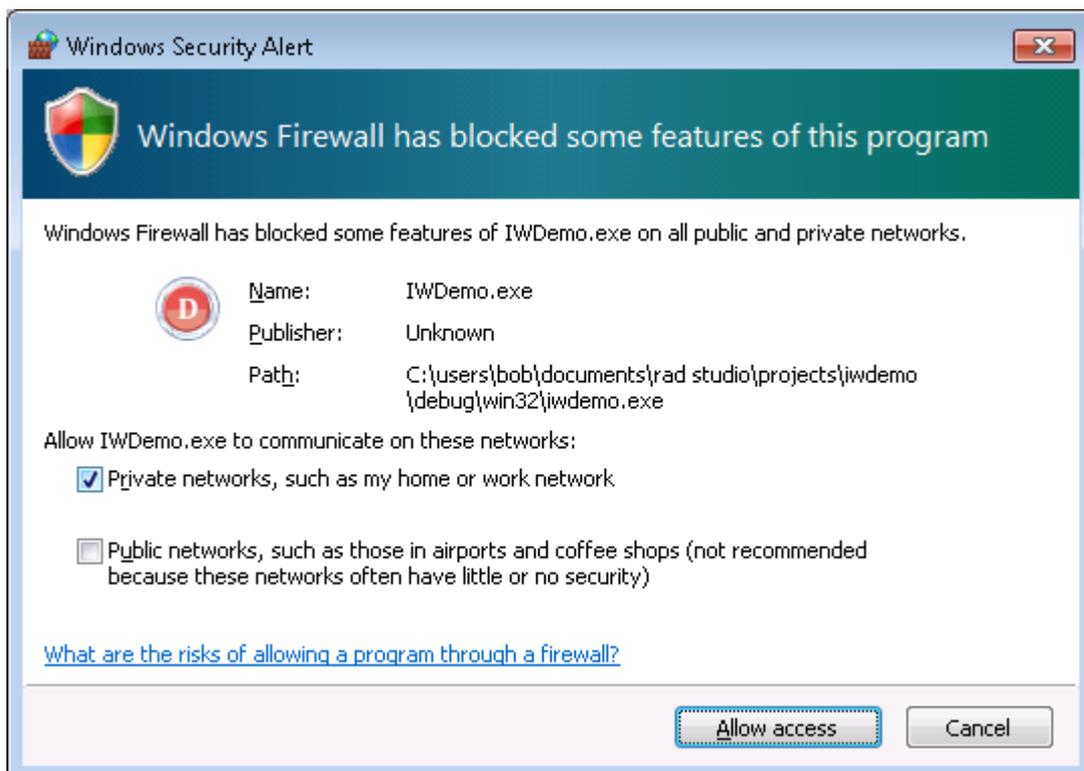
GUI Mode

The argument True specifies that we want to enforce the IWDemo application to start in GUI mode. Note that it's a default argument, and the default is False. Meaning the application should not be forced to start in GUI mode, in which case we need to pass the /GUI command-line flag to the IWDemo.exe in order to start it in GUI mode:

```
IWDemo.exe /GUI
```

The option GUI is not case sensitive, so we can also pass /gui for example.

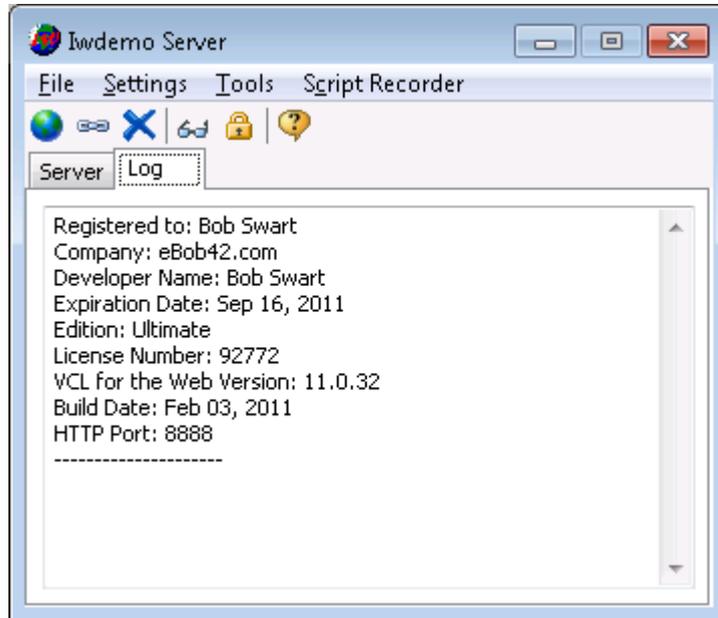
As soon as we run the IWDemo application, in GUI mode, we may get a message with a Windows Security Alert dialog, about the feature that has blocked some features of the IntraWeb application:



Obviously, we want to allow the IWDemo.exe application to continue running and operating, so click on Allow access to enabled the IWDemo executable to start the HTTP server inside it.

The IWDemo executable will show up as a little form application, with a number of options to start a specific browser from the development machine. However, we can directly connect to it from any other machine (connected to this particular development or deployment machine).

We can also see the registration information about this Ultimate version of IntraWeb XI as registered to Bob Swart.



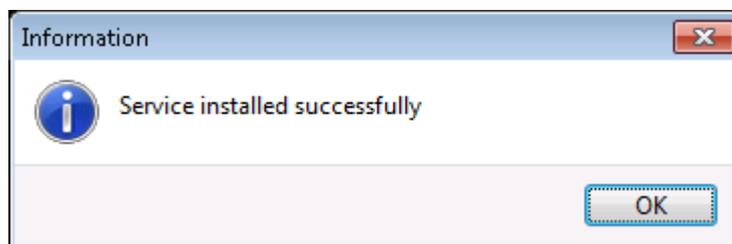
We can also see the VCL for the Web version used for the application (11.0.32 in this case) as well as the HTTP Port that the application is listening to.

Service Mode

So far, we've seen the effect of explicitly passing True to the call to TIWStart.Execute. The alternative, passing False, means that the application will be started in Service mode, which by default does nothing unless we pass the /install command line option to actually install the service:

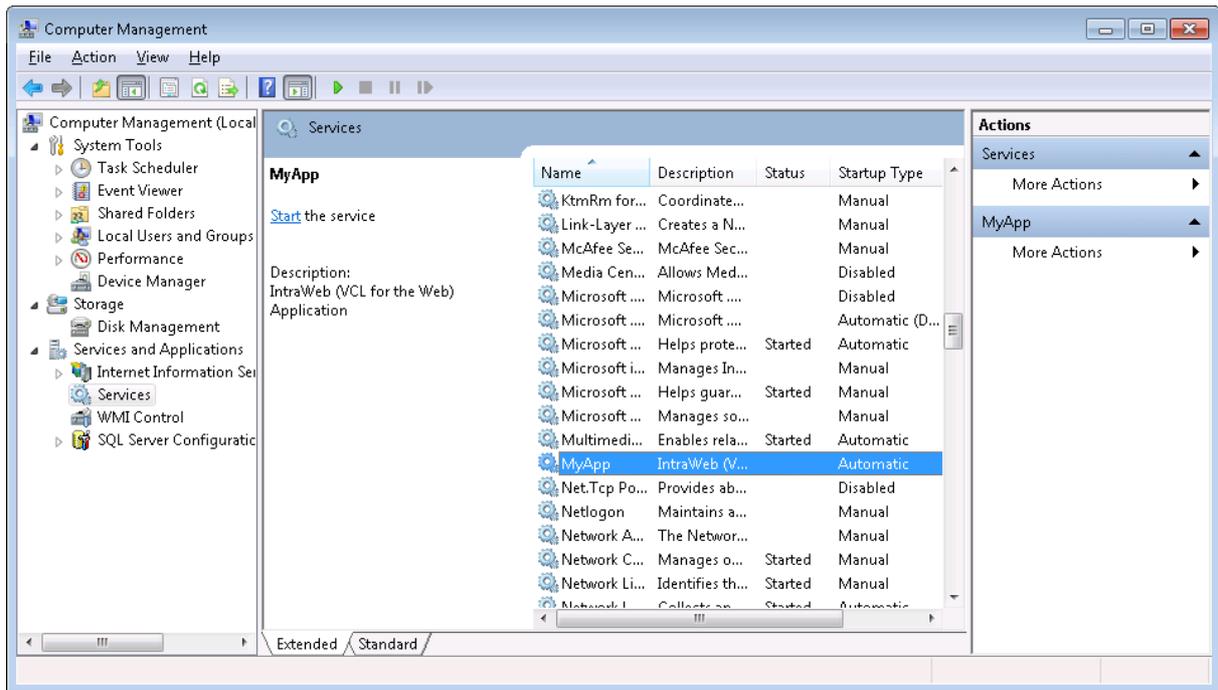
```
IWDemo.exe /install
```

Which will install the IWDemo.exe as Windows Service, but will not automatically start the service. The result is a dialog in case installing the service was successful:



If you do not see the dialog, then make sure you run the IWDemo.exe with the "Run as Administrator" option, which is required in Windows Vista and later.

In order to uninstall the service we need to call IWDemo.exe again with the /uninstall option. However, if we want to start the IWDemo service, we need to start it explicitly. Using the Computer Management Console, we can locate the MyApp IntraWeb (VCL for the Web) Application.



Note that the Startup Type is set to Automatic, so a reboot would automatically start it for us. Also note that the name MyApp is not very descriptive, so we'll change that in a moment.

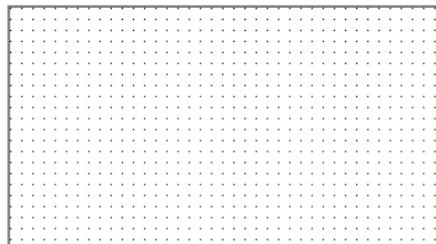
Apart from using the Computer Management Console, we can also start the service from the commandline, provided we know the name of the service (which is MyApp in this case).

```
Net start MyApp
```

And we can stop the service again using "net stop MyApp". Note that uninstalling the service will not stop it, but will delete the service as soon as it's stopped.

Server Controller

Let's now start to configure the properties of the Server Controller in unit ServerController.pas. It looks like an empty form, but is in fact more like an empty data module – we can place non-visual components on it.



The source code for the ServerController is as follows:

```
unit ServerController;
interface
uses
  SysUtils, Classes, IWebServerControllerBase, IWebBaseForm, HTTPApp,
  // For OnNewSession Event
  UserSessionUnit, IWebApplication, IWebAppForm;
```

```
type
  TIWServerController = class(TIWServerControllerBase)
    procedure IWServerControllerBaseNewSession(ASession: TIWApplication;
      var VMainForm: TIWBaseForm);
  private

  public

  end;

  function UserSession: TIWUserSession;
  function IWServerController: TIWServerController;

implementation
  {$R *.dfm}

uses
  IWInit, IWGlobal;

function IWServerController: TIWServerController;
begin
  Result := TIWServerController(GServerController);
end;

function UserSession: TIWUserSession;
begin
  Result := TIWUserSession(WebApplication.Data);
end;

procedure TIWServerController.IWServerControllerBaseNewSession(
  ASession: TIWApplication; var VMainForm: TIWBaseForm);
begin
  ASession.Data := TIWUserSession.Create(nil);
end;

initialization
  TIWServerController.SetServerControllerClass;
end.
```

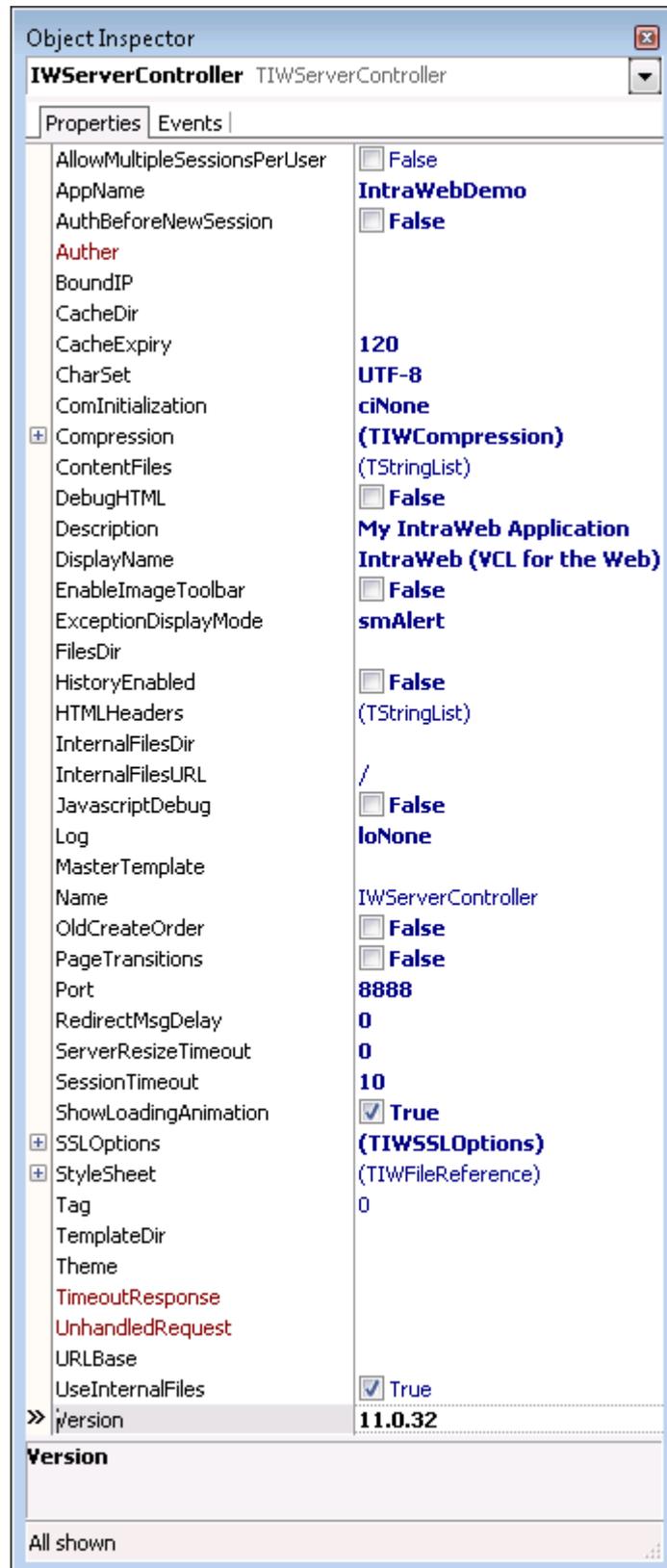
When a new session is started, then the OnNewSession event handler will make sure that a new UserSession is created, of type TIWUserSession, derived from TIWUserSessionBase:

The two global functions UserSession and ISServerController return the current user session (from the current thread) and the global server controller instance.

To configure the ServerController itself, we should view the Design of the TIWServerController and look at the properties in the Object Inspector. Start by giving the AppName property a more sensible value like IntraWebDemo (instead of MyApp). The AppName must get a unique value (i.e. a name not used by any others on the same machine) if you want to install the IntraWeb application as a Windows service.

TIWServerController Properties

Each IntraWeb application has one and only one ServerController. The ServerController has a number of properties that control the IntraWeb application.



We will examine the most important ServerController properties in some more detail.

AllowMultipleSessionsPerUser

The sessions in IntraWeb are cookie based, which means they no longer use hidden fields or the URL to display session information. This also means that a new browser tab of window shares the same IntraWeb session for a user.

If you want the browser tabs to have different user sessions, then you must use and set the AllowMultipleSessionsPerUser property to True. This will ensure that the URL will now again contain the Session ID. The session cookie is still used, but with its path set to /SessionID to allow multiple cookies to be used – each by one browser tab or window.

AppName

The AppName String property should uniquely identify your IntraWeb application. This is the name by which the IntraWeb application is registered as a Windows Service, hence the need to be unique. By default it's set to MyApp, which we saw in the screen shot of the Computer Management Console. Using MyApp as value for the AppName property is not a good idea if you build and install more than just one IntraWeb application, so make sure to change it into something else. In the screenshot on the previous page, I've already changed it to IntraWebDemo by the way.

AuthBeforeNewSession

The AuthBeforeNewSession Boolean property (set to false by default) specifies whether authentication checks should be performed before starting a new session (with data modules etc.). The default setting of AuthBeforeNewSession is false, which means that by default the authentication will take place after the session is created. This means that if you get an unauthorised attack with many login attempts, then sessions will be created for a failed authorisation request, which can be a performance hit. If you set AuthBeforeNewSession to true then the benefit is that your performance will not suffer under an intruder attack (with no unnecessary sessions), but the downside is that we cannot use the session with our data module to perform the authentication (for example in the OnCheck event of the TIWAutherEvent component).

So it's a trade-off between a potential performance hit (creating a session without knowing for sure the user can login to the system), and ease of use (being able to use the session and data module for the authentication itself).

Auther

Authentication itself is no longer done with the AuthList property or the OnBeforeAuthenticate event, but is now done – in IntraWeb XI Ultimate only - using the Auther property which can be assigned to an instance of a TIWAuther descendent class. The TIWAuther base class is defined as follows:

```
type
  TAutherPolicy = (apRestrictAll, apRestrictNone);

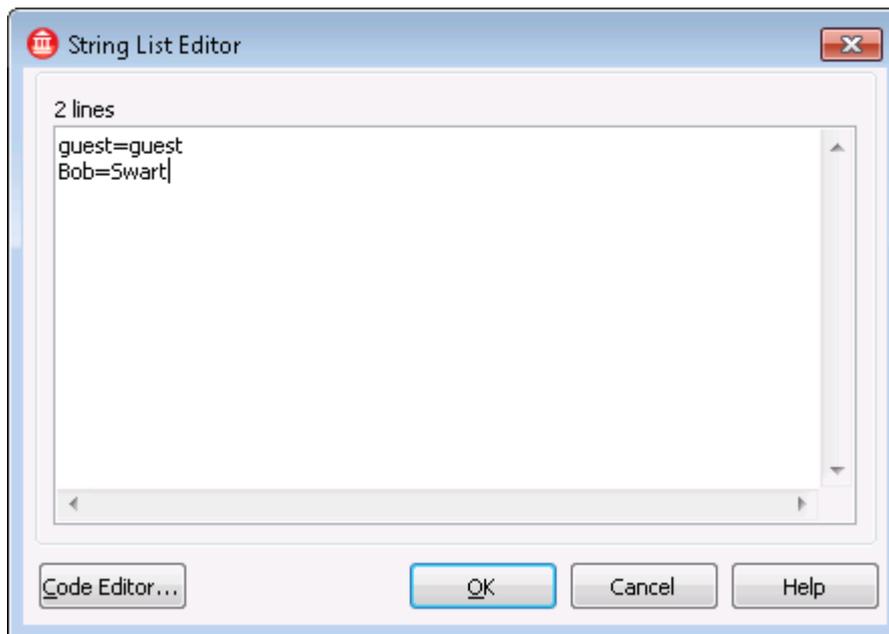
  TIWAuther = class(TComponent)
  private
    FAutherPolicy: TAutherPolicy;
    procedure SetAutherPolicy(const Value: TAutherPolicy);
  public
    function Check(const aUser: string; const aPass: string): boolean;
      virtual; abstract;
    constructor Create(AOwner: TComponent); override;
  published
    property AutherPolicy: TAutherPolicy read FAutherPolicy
      write SetAutherPolicy;
  end;
```

Using the AuthPolicy we can specify if the IWAAuther delegate should restrict it to all or to none.

There are three kinds of TIWAAuther derived components already contained with IntraWeb XI Ultimate: TIWAAutherList, TIWAAutherEvent, and TIWAAutherINI.

The **TIWAAutherList** component has the AuthPolicy property set to apRestrictAll and offers a List property of type TStrings to enter a list of usernames and passwords, similar to what the AuthList property offered in earlier versions of IntraWeb.

To allow a user guest with password guest, and a user Bob with password Swart, we can fill the following list:



The **TIWAAutherEvent** component has the AuthPolicy property set to apRestrictAll and offers an OnCheck event handler where we can implement our authentication check, for example hardcoded as follows:

```
function TIWServerController.IWAAutherEvent1Check(const aUser,  
    aPass: string): Boolean;  
begin  
    Result := (aUser = 'Bob') and (aPass = 'Swart');  
end;
```

Note that this is an event handler where we could decide to use the UserSession for example to get access to a database table. In that case, the AuthBeforeNewSession property must be set to False (the default value) to ensure that the session exists if we want to use it for the authentication.

The **TIWAAutherINI** component has the AuthPolicy property set to apRestrictAll and offers the ability to use a .ini file as source for the users and passwords. The .ini file must be called #Auth.ini (with a # as first character), and the contents can be as follows for one user Bob with password Swart:

```
[Bob]  
Password=Swart
```

Note that the passwords are stored unencrypted. But since the #Auth.ini file starts with the # character, the Integrated Page Mode feature of IntraWeb XI will not server this file so visitors will not be able to "request" the .ini file.

BoundIP

The BoundIP property specifies the IP address that the server will be bound to. Mainly useful if the IntraWeb application happens to run on a machine with more than one network card for example, and you want the IntraWeb application only to respond to incoming requests from one of these network cards.

CacheDir

This property can be used to specify the path where the IntraWeb application can store temporary files like HTML and images. You should leave it empty, so IntraWeb will determine the location of the CacheDir by itself.

If you want to control the location of the CacheDir, then be aware that the full contents of the CacheDir may be deleted by the IntraWeb application at any time. So if you place your own files in it, do not expect them to last. And certainly make sure not to point the CacheDir to a location with important subdirectories, like C:\ for example.

CacheExpiry

This property specifies how many minutes (not hours), the temporary files will be kept in the CacheDir.

CharSet

The CharSet is set to UTF-8 by default, which is the default Unicode encoding format for the internet, so there is little reason to change that.

ComInitialization

The ComInitialization property of type TComInitialization (ciNone, ciNormal, ciMultiThread) is needed when the application requires the use of (D)COM. Since IntraWeb executes requests in different threads, each thread must make its own call to CoInitialize.

By default this property is set to ciNone, to indicate that no COM support is required. If you need COM support - for example if you want to connect to a DataSnap server - then you need to set ComInitialization to ciNormal or ciMultiThreaded.

Personally, I've found no reason not to set it to ciMultiThreaded when working with COM in one way or another inside an ISAPI DLL, although IIS 6 and higher may handle the required ComInitialization already for you, and you should set it to ciNone (*but it looks like it's ignored when set to some other value, so ciMultiThread won't hurt either*).

Compression

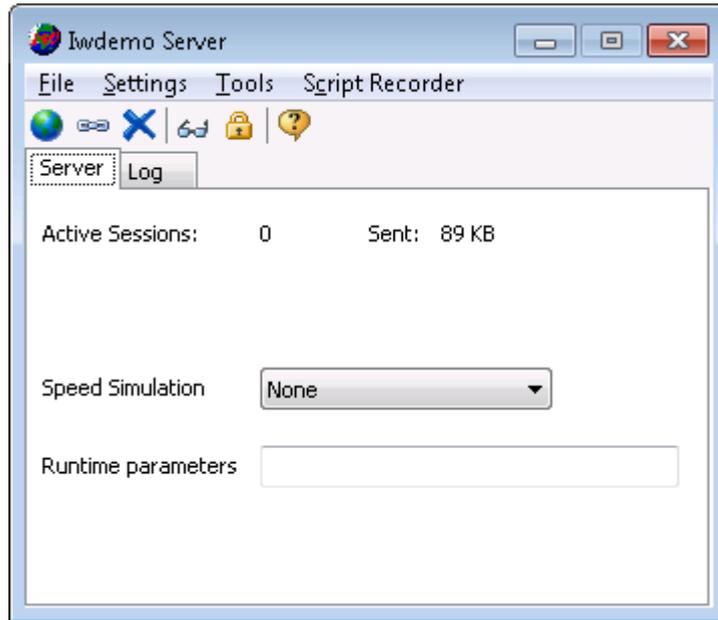
The Compression property of type TIWCompression is used to specify the level of HTML compression used by the (output of the) IntraWeb application. By default, Compression is not enabled, but we can set the Enabled subproperty to True in order to enable the HTML Compression.

The Level subproperty controls the level of the compression in a range of 0..9, the default Level is 6.

Note that in order to use compression, the zlib.dll must be deployed with your application in the same directory (even if the zlib.dll can usually be found on your system in the \Windows\System32 directory). Tests have indicated that the zlib.dll from Windows XP works fine, but the zlib.dll from Windows Vista does not work with the HTML Compression feature in IntraWeb.

You can download the zlib.dll from <http://downloads.atozed.com/intraweb/zlib.zip> to make sure you have a version that will always work with your IntraWeb applications.

In order to get a feeling for the amount of HTML that is sent over the internet connection, see the Sent information on the IntraWeb stand-alone application.



Without compression a given (large) form with a grid can produce several hundred KBs, while the same form using compression can be shrunk down to less than a hundred KB. This is of course a significant saving on the amount of HTML that is transferred over the internet.

Note that most but not all browsers may support HTML compression. But browsers that can decompress encoded content will request compressed files. Browsers that do not understand compressed content will simply request and receive the files uncompressed, not benefiting from the improved download times that content encoding compliant browsers can offer.

ContentFiles

We can use the ContentFiles property to add a list of files, especially .js and .css files, that will be added when rendering the page. Pseudo code from the TIWJQueryWidget components shows the following implementation of RenderHTML (placed in comments):

```
function TIWJQueryWidget.RenderHTML(
  AContext: TIWBaseHTMLComponentContext): TIWHTMLTag;
var
  LPageContext: TIWPageContext40;
  I: Integer;
  LFile: string;
begin
  LPageContext := TIWPageContext40(AContext.PageContext);
  for I := 0 to ContentFiles.Count - 1 do begin
    LFile := ContentFiles[i];
    if AnsiPos('javascript:', LFile) > 0 then begin
      Delete(LFile, 1, 11);
      LPageContext.AddPreScriptFile(GServerController.FilesURL + LFile);
    end else if AnsiPos('css:', LFile) > 0 then begin
      Delete(LFile, 1, 4);
      LPageContext.AddLinkFile(GServerController.FilesURL + LFile)
    end;
  end;
  LPageContext.AddToOnReady(OnReady.Text);
  Result := HTMLTag;
end;
```

DebugHTML

If you set the DebugHTML property to True, then HTML and JavaScript lines will end with a line break, which makes the source inside a browser a bit easier to read (and “debug”).

Description

The Description String property is used to contain the (internal) description of the IntraWeb application. This value is used together with the AppName and DisplayName when the IntraWeb application is registered as a Windows Service.

DisplayName

The Description String property is used to contain the DisplayName of the IntraWeb application. This value is used together with the AppName and Description when the IntraWeb application is registered as a Windows Service.

EnableImageToolbar

The EnableImageToolbar property can be used to enable the image toolbar which is shown by Internet Explorer. By default, the property is set to False, so the image toolbar will not be shown.

ExceptionDisplayMode

The EnableDisplayMode property controls how exceptions are presented to the enduser. By default, exceptions will be shown in an Alert-style JavaScript dialog box, but the alternative can be to show the exception in a new window, in the same window, or in the same window frame.

Note that this is a single setting for the entire IntraWeb application (although you are free to handle your own exceptions in a try-except block and present them to the enduser in any way you want).

FilesDir

The FilesDir property specifies the path where the application files are stored during execution. You should leave this property unassigned, unless you have a good reason to change the default location.

HistoryEnabled

The HistoryEnabled property is set to false by default, and should be left alone. It controls whether the browser should maintain history information, and the back and forward buttons (which should not be used in combination with IntraWeb applications due to the state decoded in the URL).

HTMLHeaders

The HTMLHeaders property can be used to insert additional HTML headers that will appear in the <head>...</head> section of the generated HTML document. This is related to the ContentFiles property if you want to insert CSS or JS contents directly into the header of the generated page.

InternalFilesDir

The InternalFilesDir property has only effect for ISAPI applications, and specifies the physical directory where internal files like JavaScript source files are loaded from. This property is ignored in a stand-alone IntraWeb application.

InternalFilesURL

The InternalFilesURL property has only effect for ISAPI applications, and specifies the URL where internal files like JavaScript source files are loaded from. This property is ignored in a stand-alone IntraWeb application. The default value is / or the root of the website.

JavascriptDebug

The JavascriptDebug property can be set to True (default is False). It controls the value of the IWDEBUG global JavaScript variable that can be used to force JavaScript messages when debugging:

```
</style><script type="text/javascript">
  var IWDEBUG = false;</script>
<script type="text/javascript" src="/$/js/IWPreScript.js_11.0.32"></script>
```

When JavascriptDebug is set to True, then the value of IWDEBUG will be set to true as well.

Log

The Log property of the ServerController can be set to loNone (the default value) or loFile. With a value of loFile, a log file will be generated on disk.

MasterTemplate

The MasterTemplate property of the ServerController can be used – in combination with Layout Managers - to define a template that will be used for all forms in the IntraWeb application.

PageTransitions

The PageTransitions property of the ServerController controls whether or not pages can fade during the transitions. The default value is False.

Port

The Port property specifies the Port that the IntraWeb stand-alone application or Windows Service will use (and will listen to). Note that you must ensure that your firewall allows traffic through this specific port, or nobody will be able to reach your IntraWeb server.

The value of this property has no effect for an ISAPI DLL, since IIS will control the port (typically 80 for HTTP and 443 for HTTPS connections).

For IntraWeb standalone or service applications, you must ensure that the Port property is a unique value, and not used by another application, otherwise you will get an error message (only one server application can listen to a port at a single time).

If you do not know if your firewall is “open” to allow customers to connect to your IntraWeb stand-alone application, I can recommend the ShieldsUP! service from Gibson Research at <http://www.grc.com> which will test your firewall (for internet access).

RedirectMsgDelay

The RedirectMsgDelay property specifies the number seconds before the message with an indication for a redirect to an previous correct state or an error page takes place (for example if the user pressed the Back button).

ServerResizeTimeout

When you resize a browser window that runs an IntraWeb application, the new size of the browser window is sent to the server. Using the `ServerResizeTimeout` property you can specify how many seconds the browser will wait before the new size of the browser window will be sent to the server. The default is 0, which means an immediate submit after the resizing stops, and after each small resize thereafter, which will not work good for slow connections.

SessionTimeout

The `SessionTimeout` Integer property specifies the minutes of inactivity before a user's session object is destroyed. This can happen if the user has closed the browser window, for example, and can no longer communicate with the same session anymore. By default, `SessionTimeout` is set to 10 minutes. For testing purposes it's sometime useful to set `SessionTimeout` to 1 (note that 0 has the same effect as setting it to 1, and if you're using cookies for session information, then a value of 0 means that a cookie is a session cookie, and won't be stored on your clients' machines).

ShowLoadingAnimation

The `ShowLoadingAnimation` boolean property controls the small animation that is displayed while your IntraWeb form is being loaded. By default, this property is set to true, so the animation will be shown.

SSLOptions

The `SSLOptions` `TIWSSLOptions` property can be used to specify SSL Options, and consist of a number of subproperties:

The `SSLOptions.CertificatePassword` String property specifies the password to use to access the SSL certificates.

The `SSLOptions.NonSSLRequest` `TIWNonSSLRequest` property can be used to handle a non-SSL request and can be set to `nsAccept`, `nsBlock` or `nsRedirect`.

The `SSLPort` property contains the SSL port to list on for HTTPS requests.

The `SSLVersion` property contains the SSL version, which can be `sslv2`, `sslv23`, or `sslv3`.

StyleSheet

The `StyleSheet` String property specifies the Filename or URL that contains the default stylesheet for the IntraWeb application.

TemplateDir

The `TemplateDir` String property specifies the path where the HTML template files must be placed in that are used by the `TemplateProcessors`. The property is related to the `MasterTemplate` property of the `ServerController` and the

TimeoutResponse

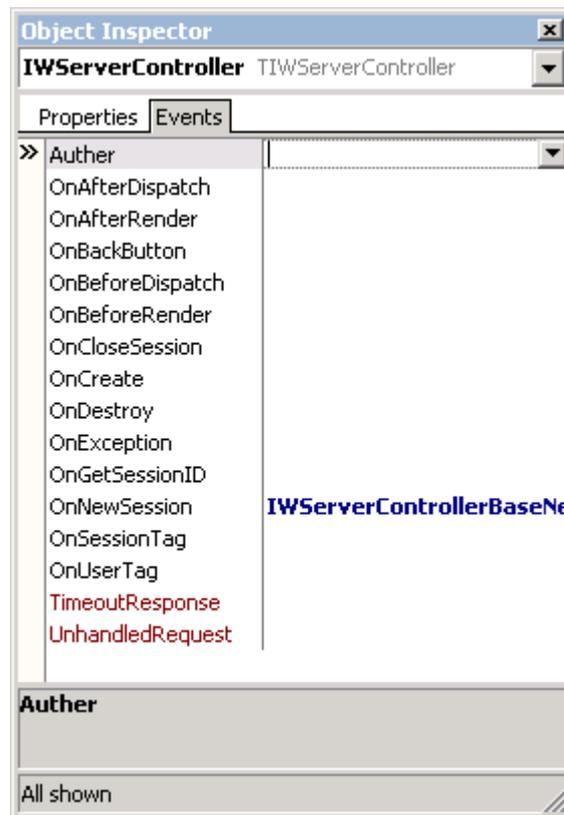
The `TimeoutResponse` property replaces the old `SessionTimeoutURL` property of the `ServerController`. We must now use a URL Responder to respond to a time out message if we want to.

There are three built-in URL Handlers in IntraWeb XI. The `TIWURLResponderRedirect` can be used to redirect to another URL. Alternatives include `TIWURLResponderEvent` to use an event handler to handle the request and produce a response for the timeout event, and a `TIWURLResponderDirListing` to provide a directory listing.

For the old behaviour to work, we should place a `TIWURLResponderRedirect` on the `ServerModule`, assign a value to the URL property, and assign the responder to the `TimeoutResponse` property of the `ServerController`.

TIWServerController Events

The ServerController also has a number of useful events that can be used by the IntraWeb application.



OnAfterDispatch

The OnAfterDispatch event handler is called right after the action is dispatched by the server. This event handler can be used in combination with the OnBeforeDispatch.

```
procedure TIWServerController.IWServerControllerBaseAfterDispatch(  
    Sender: TObject; Request: TWebRequest; Response: TWebResponse;  
    var Handled: Boolean);  
begin  
  
    end;
```

OnAfterRender

The OnAfterRender event handler is called after the rendering has been done.

```
procedure TIWServerController.IWServerControllerBaseAfterRender(  
    ASession: TIWApplication; AForm: TIWBaseForm);  
begin  
  
    end;
```

OnBackButton

The OnBackButton doesn't fire when the back button is pressed, but only after an inconsistent state is detected (if you write any code for this event handler, then HistoryEnabled should be set to True and ShowResyncWarning to False so you have to handle the resync options yourself in the OnBackButton event handler):

```
procedure TIWServerController.IWServerControllerBaseBackButton(  
  ASubmittedSequence, ACurrentSequence: Integer; AFormName: String;  
  var VHandled, VExecute: Boolean);  
begin  
  VHandled := True;  
  VExecute := true;  
  if WebApplication.FindComponent(AFormName) <> nil then  
    WebApplication.SetActiveForm(WebApplication.FindComponent(AFormName));  
  else  
    if AFormName = 'IWFrMMain' then  
      WebApplication.SetActiveForm(TIWFrMMain.Create(WebApplication));  
  end;  
end;
```

The code in the event handler above tries to find an existing instance of the form that the browser was showing, and if so, reactivates that form (in the current - resync - state). If not, then we can use the AFormName argument to find create a new instance of the required form and show it instead.

I still think it's better not to use HistoryEnabled, ShowResyncWarning and OnBackButton, and just convince the enduser that they are not supported in web applications (also called weblications).

OnBeforeDispatch

The OnBeforeDispatch event handler can be used to examine the incoming request, and potentially to write your own response and mark the request as handled.

```
procedure TIWServerController.IWServerControllerBaseBeforeDispatch(  
  Sender: TObject; Request: TWebRequest; Response: TWebResponse;  
  var Handled: Boolean);  
begin  
  
end;
```

OnBeforeRender

The OnBeforeRender event handler is called before rendering takes place – the counterpart of the OnAfterRender event handler.

```
procedure TIWServerController.IWServerControllerBaseBeforeRender(  
  ASession: TIWApplication; AForm: TIWBaseForm; var VNewForm: TIWBaseForm);  
begin  
  
end;
```

OnCloseSession

The OnCloseSession event handler is fired when a session is closed. The only argument ASession is of type TIWApplication - the IntraWeb application variable.

```
procedure TIWServerController.IWServerControllerBaseCloseSession(  
  ASession: TIWApplication);  
begin  
  ASession ...  
end;
```

OnException

The OnException event handler is called when an exception is not handled by the IntraWeb application. If we don't implement OnException, then the following default code is executed:

```
procedure TIWServerController.IWServerControllerBaseException(  
  AApplication: TIWApplication; AException: Exception);  
begin  
  AApplication.ShowMessage(AException.Message);  
end;
```

Personally, I think it would make sense to do something special here, not just show the exception, but also log the exception or do something useful with it (note: the procedure Log doesn't exist, but I'm just showing it here as an example – feel free to use a tool like CodeSite or provide your own logging tool).

```
procedure TIWServerController.IWServerControllerBaseException(  
  AApplication: TIWApplication; AException: Exception);  
begin  
  CodeSite.Send(AException.ClassName + ': ' + AException.Message);  
  AApplication.ShowMessage(AException.ClassName + ': ' + AException.Message);  
end;
```

Apart from logging the exception, it also helps to log the user and other information, but the code above is just an example to show you where to place your own IntraWeb exception and error handling code.

See the TIWApplication.ShowMessage section for all options that you can use when showing messages (for example in a alert pop-up dialog, a new browser window, or the current window).

OnGetSessionID

The OnGetSessionID event handler is used to return the session ID:

```
procedure TIWServerController.IWServerControllerBaseGetSessionID(  
  ASession: TIWApplication; var VNewSessionID: string);  
begin  
  
end;
```

OnNewSession

The OnNewSession event handler is called when a new user session is created. The default code creates the enduser session object as follows:

```
procedure TIWServerController.IWServerControllerBaseNewSession(  
  ASession: TIWApplication; var VMainForm: TIWAppForm);  
begin  
  ASession.Data := TUserSession.Create(nil);  
end;
```

Apart from the IntraWeb application variable, we also get the main form (the first one that will be shown) as argument to this event handler.

If you create special things here, or perform some operations (like a unique logfile for each specific session), then the OnCloseSession is the obvious place to undo these special things again.

IntraWeb TIWApplication Properties

Before we move on to the second generated unit, with the IntraWeb Application Form of type TIWAppForm, let's first examine in some detail the IntraWeb Application object itself of type TIWApplication.

Each enduser (connected via browser or other device) will get its own session, which in fact will be a unique, multi-threaded, instance of TIWApplication.

ActiveForm

The TIWApplication.ActiveForm property of type TComponent (and not of the expected TIWBaseForm type) points to the enduser's current active IntraWeb Application Form.

ActiveFormCount

The TIWApplication.ActiveFormCount Integer property contains the number of Active Application Forms. I expect this to be either 0 (if no form is shown) or 1 (if one form is shown).

AppID

The TIWApplication.AppID String property contains a unique string that identifies the enduser's session. You can display this string for tracing/debugging purposes.

ApplicationURL

The ApplicationURL String property returns the URL for the application.

Browser

The TIWApplication.Browser TIWBrowser property contains the detected value of the browser that the enduser is using. Possible values can be brUnknown, brIE, brNetscape6, brOpera, brNetscape4, brOther and brHTML32Test (for internal testing purposes).

Data

The TIWApplication.Data TObject property is used to maintain state and session information. It is a pointer that is assigned to point to an object of type TUserSession (in the OnNewSession event handler) which is derived from TComponent. The object connected to the Data pointer is automatically destroyed when the session ends.

FormAction

The TIWApplication.FormAction String property specifies the value of the Action of the current Active IntraWeb Application Form.

FormCount

The TIWApplication.FormCount Integer property contains the number of IntraWeb Application Forms currently in memory.

Note that this number also includes the number of data modules.

Forms

The TIWApplication.Forms property contains a list (array) of all IntraWeb Application Forms and data modules that have been created and are still available in memory (i.e. have not been Released, yet).

IP

The TIWApplication.IP String property contains the IP address of the enduser that started the request currently being handled. Can be useful for logging or security purposes.

IsCallback

We can ask the IntraWeb application if the current request is part of a callback method.

LastAccess

The TIWApplication.LastAccess TDateTime property contains a timestamp of the last access by the current user - the value of this timestamp in combination with the TIWServerControllerBase.SessionTimeout property can determine when the session will time out.

RedirectURL

The TIWApplication.RedirectURL String property contains the URL that will be used if the enduser in the browser will be sent to after the TerminateAndRedirect method is called.

ReferringURL

The `TIWApplication.ReferringURL` String property contains the URL that referred to the IntraWeb application. The value is retrieved from the HTTP Referrer header.

Request

The `TIWApplication.Request` `TWebRequest` property contains the incoming HTTP request of type `TWebRequest`. This is the good-old Request that we've known since `WebBroker`, and we can use `Request.ContentFields` (for form POSTs), `Request.QueryFields` (for form GETs) as well as `Request.CookieFields` (for cookies) to get request input when required.

Response

The `TIWApplication.Response` `TWebResponse` property contains the outgoing HTTP response of type `TWebResponse`. Although this is also the good-old Response that we've known since `WebBroker`, it's probably not recommended to write output to this object. I'll have to perform some more experiments to find out what the result can be...

RunParams

The `TIWApplication.RunParams` `TStrings` property contains a list of strings with the HTTP variables that were passed as part of the application's URL.

SecureMode

The `SecureMode` property specifies whether or not a secure mode (like HTTPS) is used for the application.

SessionTimeout

The `SessionTimeout` property contains the number of minutes before the session will be timed out.

Terminated

The `TIWApplication.Terminated` Boolean property is True if the IntraWeb web application is correctly terminated using the `Terminate` or `TerminateAndRedirect` function, which are defined as follows:

```
procedure Terminate(const AMsg: String);
procedure TerminateAndRedirect(const AURL: String); overload;
procedure TerminateAndRedirect(const AURL: string; const AMsg: string); overload;
```

The `AMsg` in `Terminate` is a message that can be passed by the application and will be shown to the enduser in the browser, for example:

```
WebApplication.Terminate('Goodbye ' + WebApplication.AuthUser)
```

The `AURL` of `TerminateAndRedirect` is the URL to which the browser will be redirected.

TerminateMessage

The `TIWApplication.TerminateMessage` String property is a read-only property that contains the message that will be displayed to the enduser if the application is terminated, and something was passed in the `AMsg` argument to `Terminate`.

You can override this message by calling the `Terminate` method with the `Msg` argument.

TerminateURL

The `TIWApplication.TerminateURL` String property specifies the URL that the enduser will be redirected to if the IntraWeb application was terminated correctly using the `TerminateAndRedirect` method.

TrackID

The `TIWApplication.TrackID` Cardinal property is used as session tracking marker to keep the user in sync if they browse their history or use the back button and try to post old data.

UserCacheDir

The `TIWApplication.UserCacheDir` specifies the user specific cache that holds files that are created and kept for the life of the user session. Graphics and other resources that are common among forms but are specific to a user are stored here.

TIWApplication Methods

The `TIWApplication` class has no events, but a number of methods that we can call from the `WebApplication` object. Not all events are useful, or meant to be called directly, so this list is limited to those methods that make sense to be called by IntraWeb developers.

GoToURL

The `TIWApplication.GoToURL` method redirects the browser to the specified URL.

```
procedure GoToURL(const AURL: String);
```

MarkAccess

The `TIWApplication.MarkAccess` method updates the `LastAccess` field with the current time. You can use this for testing/debugging purposes to prevent the session from timing out, for example.

SendFile

The `TIWApplication.SendFile` method returns a file to the user (received by the browser).

```
procedure SendFile(const APathname: String; AContentType: String = '';
                  AFilename: String = '';
                  const AAttachment: Boolean = False);
```

If `AContentType` is omitted or an empty string then the browser will retrieve the content type from the operating system. If `AFilename` is omitted or an empty string then the filename will only take `APathname`. If `AAttachment` is set to `True`, then the `ContentType` will be ignored (`AAttachment` and `AContentType` are mutually exclusive).

SendStream

The `TIWApplication.SendStream` method returns a data stream to the user. The stream is freed after it has been transmitted, so we should only create the stream but not free it after calling `SendStream`.

```
procedure SendStream(AStream: TStream;
                   const AContentType: String = '';
                   const AFilename: String = '';
                   const AAttachment: Boolean = False);
```

If `AContentType` is omitted or an empty string will retrieve the content type from windows. If `AAttachment` is `True`, then the `Content-Type` will not be taken into consideration, i.e. `AAttachment` and `AContentType` are mutually exclusive.

ShowMessage

The `TIWApplication.ShowMessage` method displays a message to the user and offers them an "OK" button. After viewing the message and clicking the OK button they are returned back to the previous form.

```
procedure ShowMessage(Amsg: String;
                    const AType: TIWShowMessageType;
                    ATemplate: String);
```

The AMsg String is the message to be displayed to the enduser in the browser. The message can be displayed in different ways, however, which is controlled by the AType argument of type TIWShowMessageType, which can be one of the following values:

smNewWindow displays the message in a new pop-up window.

smSameWindow displays AMsg in the same window and replaces the current page with AMsg. Clicking OK returns to the previous window with the previous page again.

smSameWindowFrame works the same as smSameWindow, but in this case AMsg is displayed in a scrollable frame on the page.

smAlert displays a popup modal dialog to the user.

The ATemplate String allows us to specify a template to use as well. For the format of the template see the IntraWeb Manual. If no template is specified, IntraWeb will attempt to use the IWShowMessage.html template.

Templates are ignored when AType = smAlert or AType = smNewWindow.

Terminate

The TIWApplication.Terminate method terminates the application and shows a message.

```
procedure Terminate(const AMsg: String);
```

TerminateAndRedirect

The TIWApplication.TerminateAndRedirect *overloaded* method terminates the IntraWeb application and redirects the enduser to another URL.

```
procedure TerminateAndRedirect(const AURL: string); overload;
```

```
procedure TerminateAndRedirect(const AURL: string;  
  const AMsg: string); overload;
```

The last form of TerminateAndRedirect will first show the AMsg message, and then redirects the user to the AURL. This is done with the following HTML body:

```
<HTML>  
<META HTTP-EQUIV="Refresh" CONTENT="2;URL=http://www.ebob42.com">  
<BODY>TerminateAndRedirect</BODY>  
</HTML>
```

Where the AURL is placed in the URL part, and the AMsg is placed in the body tags. The timeout is set to 2 seconds, and I haven't found a way to overrule that, yet.

TIWAppForm

Time to take a closer look at the IntraWeb Form in IWFormMain.pas. This form is derived from TIWAppForm and produces HTML. We can use all component from the IW Standard, IW Data, and IW Control categories, plus the IW URL Responders (with the TIWURLResponderEvent, TIWURLResponderRedirect, and TIWURLResponderDirLister), and for the IntraWeb XI Ultimate and evaluation editions the IW Authentication category with the TIWAutherList, TIWAutherINI and TIWAutherEvent components.

IntraWeb TIWPageForm

Before we continue, let's examine in some more detail the properties and events of the IntraWeb Page Form (in IWformMain.pas), derived from the TIWPageForm class. There are a number of interesting properties that we should take a look at (note that not all properties have meaning, such as Cursor):

ActiveControl

The TIWForm.ActiveControl TIWControl property is used to specify the control that will get the focus when the IntraWeb Page Form is first shown in the browser.

Background

The TIWForm.Background TIWBackground property specifies the background image for the IntraWeb Page Form. We can specify a Filename or URL subproperty, and the Fixed subproperty specifies whether the background image should scroll or not.

ExtraHeader

The TIWBaseForm.ExtraHeader TStrings property can be used to specify a number of extra HTML strings that will be inserted in the <head>...</head> section of the generated HTML document.

HandleTabs

The TIWForm.HandleTabs Boolean property generates special JavaScript code to properly handle the tab key in Netscape 4 (in pages without using templates).

HiddenFields

The TIWBaseForm.HiddenFields TStrings property contains a list of hidden fields that will appear in contents of the generated HTML document.

JavaScript

The TIWForm.JavaScript TStrings property can be used to enter some additional JavaScript code that will be included in the generated HTML document.

LayoutMgr

The TIWBaseForm.LayoutMgr TIWLayoutMgrBase property specifies the specific layout manager to use.

LinkColor

The TIWBaseForm.LinkColor TIWColor property defines the colour of the links on the generated HTML document inside the browser. See also the VLinkColor property for the colour of the visited links on the generated HTML document.

ShowHint

The TIWForm.ShowHint Boolean property specifies if the fly-over hints should be shown for the IntraWeb controls - just as in "normal" Delphi applications.

StyleSheet

The TIWPageForm.StyleSheet TIWFileReference property specifies a stylesheet to use. If the StyleSheet starts with <http://> it will be treated as an URL, otherwise it is assumed to be a filename (using a fully qualified path, or present in the files subdirectory).

SupportedBrowsers

The TIWBaseForm.SupportedBrowsers TIWBrowser property specifies the browsers that will be supported by this IntraWeb application.

TextColor

The TIWBaseForm.TextColor TIWColor property specifies the colour of normal text on the web page. Note that the IntraWeb controls can specify their own colour to override this.

Title

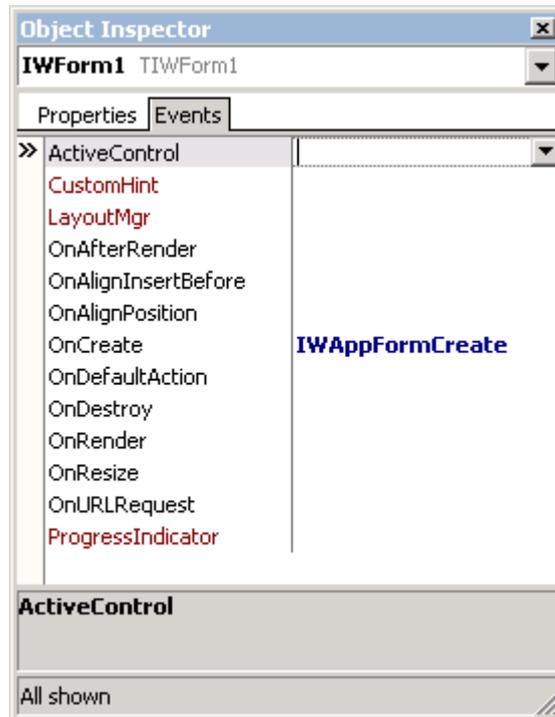
The TIWBaseForm.Title String property specifies the title for the web page. Most browsers display this in the browser window title bar. Note that you should not use the Caption property for this (which is Windows GUI only), but the Title property instead.

VLinkColor

The TIWBaseForm.VLinkColor TIWColor property specifies the colour of visited links on the IntraWeb Page Form. See also the LinkColor property.

TIWPageForm Events

The TIWPageForm has a number of events that we can hook into and write code for.



OnAfterRender

The TIWPageForm.OnAfterRender event handler is called after a IntraWeb Page Form has been rendered.

OnCreate

The TIWPageForm.OnCreate event handler is called when a new IntraWeb Page Form has been created.

OnDefaultAction

The TIWPageForm.OnDefaultAction event handler is fired if the user causes a default submit on the IntraWeb Page Form. This is usually caused by pressing the enter key in an edit field.

OnDestroy

The TIWPageForm.OnDestroy event handler is called just before an IntraWeb Page Form is destroyed (so we can clean-up what we have created in the OnCreate event handler).

OnRender

The TIWPageForm.OnRender event handler is called each time the IntraWeb Page Form is generated for display in the browser. Note that sometimes this may be called without the user actually clicking on buttons or links if they refresh the page using the refresh function in their browser.

We can use the OnRender method to add content to the web page, for example by adding items to the HiddenFields property (something which we'll do in a moment).

Designing IntraWeb Page Form

Let's start by placing a TIWLabel component from the IW Standard tab, set the Align property to alTop and the text allignment to taCenter, and give the Caption property the value "IntraWeb XI".



After you've saved your code, the source of the IWFormMain.pas unit should be as follows:

```

unit IWFormMain;
interface
uses
  Classes, SysUtils, IWAppForm, IWApplication, IWColor, IWTypes;

type
  TIWForm1 = class(TIWAppForm)
    IWLabel1: TIWLabel;
  public
    end;
implementation

  {$R *.dfm}

end.

```

Note the call to TIWForm1.SetAsMainForm in the initialization section that ensures that this TIWForm1 (derived from TIWAppForm) will be used as main form is for HTML 4 browser clients.

IW Standard Controls

IntraWeb comes with a large number of controls that can be found in the Tool Palette (but only when an IntraWeb Page is active in the designer, otherwise the controls are hidden).

The IW Standard Controls can be used in regular IntraWeb Application or Page Forms, and include 37 IntraWeb controls: TIWSilverlight, TIWSilverlightVideo, TIWApplet, TIWButton, TIWCheckBox, TIWComboBox, TIWEdit, TIWFile, TIWFlash, TIWHRule, TIWImage, TIWImageFile, TIWImageButton, TIWList, TIWLabel, TIWListBox, TIWLink, TIWMemo, TIWMenu, TIWProgressBar, TIWRadioGroup, TIWRectangle, TIWRegion, TIWText, TIWTimer, TIWGrid, TIWTreeView, TIWURL, TIWURLWindow, TIWActiveX, TIWMPEG, TIWQuickTime, TIWCalendar, TIWOrderedListbox, TIWTabControl, TIWTimeEdit, and TIWRadioButton.

I will cover their most important properties and events here, but won't cover the Async (AJAX) related properties here – these will be mentioned in the AJAX specific section.

TIWApplet

The TIWApplet component enables you to include Java applets with your IntraWeb application, or at least to "integrate" a Java applet

The TIWApplet.AppletName String property specifies the name for the Java applet. This may be useful for inter-applet communication.

The TIWApplet.AltText String property is the alternate text that should be used for this applet. This text is used as a hint, while the applet is loading, or is displayed instead of the applet when the applet cannot be loaded.

The `TIWApplet.Archive` String property specifies the name of the archive from which the applet is to be loaded.

The `TIWApplet.ClassFile` String property specifies the name of the file from which the applet is to be loaded.

The `TIWApplet.CodeBase` String property specifies the code base for the applet. If `CodeBase` is an empty string, the applet will be searched in the files directory.

The `TIWApplet.HorizSpace` Integer property specifies the size in pixels for the horizontal gutter of the applet.

The `TIWApplet.Params` TStrings property contains a list of additional parameters that are passed to the applet, in the format `name=value`.

The `TIWApplet.VertSpace` Integer property specifies the size in pixels for the vertical gutter of the applet.

TIWButton

The `TIWButton` component shows a button that can display a text and that the enduser can click on.

The `TIWButton.ButtonType` `TIWButtonType` property specifies the HTML type of the button, which can be `btSubmit`, `btReset` or `btButton` (for a Normal button). Submit buttons cause the action of the current form to be executed, reset buttons clear all the editable fields of a HTML form while normal buttons don't have any default action. Setting the `ButtonType` property changes the way the button interacts with the containing form in the rendered HTML code.

The `TIWButton.Caption` `TCaption` String property is the text that will be displayed on the button.

The `TIWButton.Color` `TIWColor` property specifies the background colour for the button.

The `TIWButton.Confirmation` String property can be used to display a confirmation prompt to the user to confirm the action of the button when the user clicks on it.

For example, if the `Confirmation` property contains *"Are you sure?"* the user will be asked this question when they click on the button with two options: confirm and cancel. If the user clicks on confirm then the `OnClick` event of the button will fire. If the user clicks on cancel the `OnClick` event will not fire.

To disable confirmation, set the `Confirmation` property to an empty string.

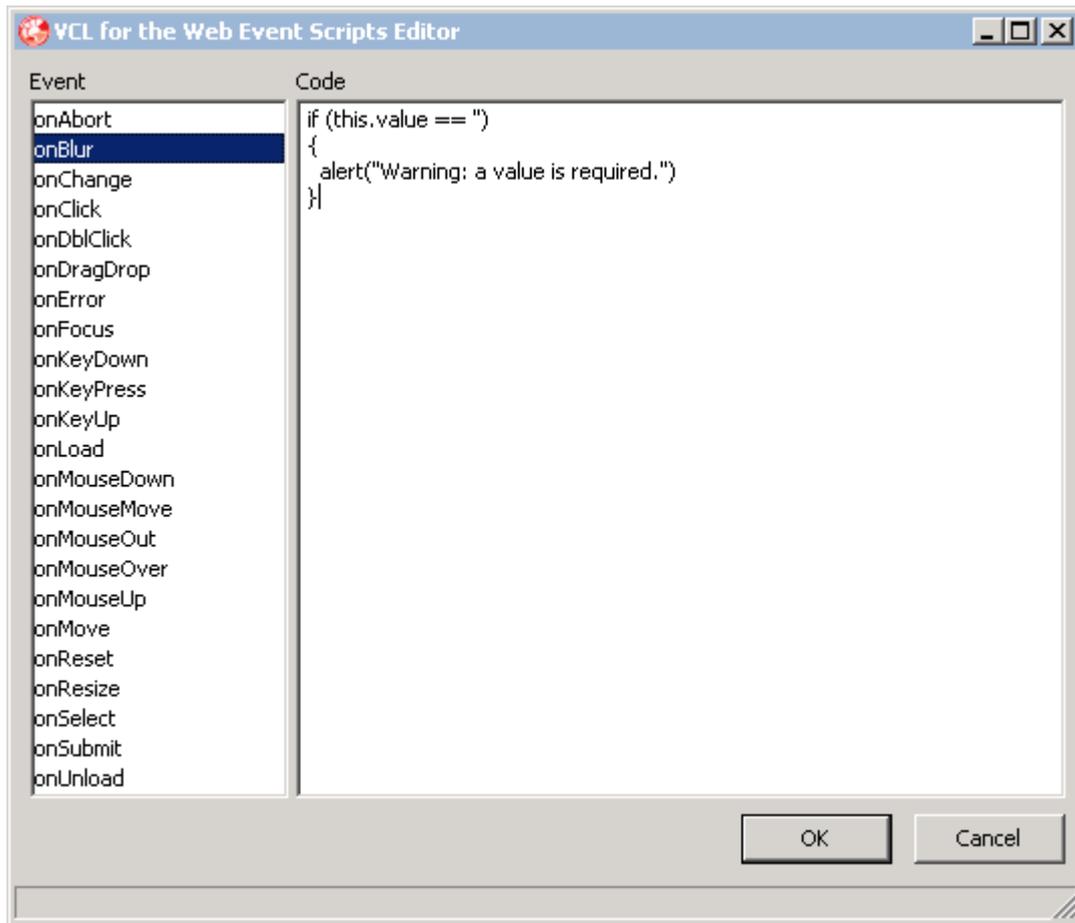
The `TIWButton.DoSubmitValidation` Boolean property specifies whether or not validation is performed by other controls when the control is submitted. Validation is done by properties such as `Required`, etc.

The `TIWButton.Enabled` Boolean property specifies whether the button should be enabled for the user or not. A disabled button will not respond to mouse clicks and not fire `OnClick` events (but it doesn't look disabled).

The `TIWButton.HotKey` String property specifies which shortcut key to use for pressing that button. The value of the `HotKey` property must be a single character (so why not make it of type `Char` instead of `String`?). The shortcut is formed as a combination of `ALT` and the value of the `HotKey` property.

The `TIWButton.ScriptEvents` `TIWScriptEvents` property can be used to add Javascript to control events to be processed on the client side. Scripts must be a Boolean evaluation or a function that returns a Boolean. If it evaluates to true, IntraWeb will continue processing the event itself if it uses the particular event. If it returns false, IntraWeb will not process the event further.

Using the IntraWeb Event Scripts Editor you can write JavaScript for `onAbort`, `onBlur`, `onChange`, `onClick`, `onDblClick`, `onDragDrop`, `onError`, `onFocus`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onLoad`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `onMove`, `onReset`, `onResize`, `onSubmit` and `onUnload` events:



These ScriptEvents can be used with a lot of the IntraWeb component, but be aware that not all of the available events are equally useful for all IntraWeb components. Example scripting code (for the OnBlur event of a TIWEdit for example) is as follows:

```
if (this.value == '')  
{  
  alert('Warning: a value is required.')}
```

The TIWButton also has an OnClick event handler that can be used to write server-side code that executes when the user clicks on the button.

TIWCheckBox

The TIWCheckBox component displays a checkbox that can also display a text, and that the user can check or uncheck.

The TIWCheckBox.Caption String property is the text that is displayed on the to the right of the checkbox in the browser.

The TIWCheckBox.Checked Boolean property specifies whether or not the checkbox is checked.

The TIWCheckBox.Confirmation String property can be used to display a prompt to the user to confirm the OnClick action of the checkbox when the user clicks on it.

The TIWCheckBox.DoSubmitValidation Boolean property specifies whether or not validation is performed by other controls when the control is submitted.

The TIWCheckBox.Editable property specifies whether the TIWCheckBox should respond to user actions or not. A disabled checkbox (Editable is set to false - and not Enable as the on-line help says) will not respond to mouse clicks, and the user won't be able to interact with the checkbox.

The `TIWCheckBox.ScriptEvents TIWScriptEvents` property can be used to add Javascript to control events to be processed on the client side. Scripts must be a Boolean evaluation or a function that returns a Boolean. If it evaluates to true, IntraWeb will continue processing the event itself if it uses the particular event. If it returns false, IntraWeb will not process the event further.

The `TIWCheckBox.Style TIWCustomCheckBoxStyle` property specifies how the checkbox will appear in the browser. It can be set to `stNormal` or `stCool` (for a graphical checkbox with colours).

The `TIWCheckBox` has an `OnClick` event handler that can be used to write server-side code that executes when the user clicks on the checkbox. Warning: don't use this event handler too often, since it means that for every `OnClick` event, a trip to the IntraWeb application has to be made.

There is also an asynchronous way to send requests to the server, offered by the `OnAsyncClick`, `OnAsyncDoubleClick`, `OnAsyncEnter` and `OnAsyncExit` events, but these will be covered later in the AJAX section.

TIWComboBox

The `TIWComboBox` component displays a combobox, with string items that we can add to the `Items` property.

The `TIWComboBox.ItemIndex Integer` property specifies the currently selected item. A value of -1 specifies that no item is currently selected.

The `TIWComboBox.Items TString` property contains the string items in the `TIWComboBox`.

The `TIWComboBox.Sorted Boolean` property specifies the sorting of items. Setting `Sorted` to `True` causes the items to be sorted alphabetically. If the property is set to `False`, the items are presented as they are provided in the `Items` property.

The `TIWComboBox.DoSubmitValidation Boolean` property specifies whether or not validation is performed by other controls when the control is submitted.

The `TIWComboBox.Editable` property specifies whether the `TIWComboBox` should respond to user actions or not.

The `TIWComboBox.ItemsHaveValues Boolean` property can be set to `True` if the items are of the format: `Name=Value`. The `Value` will be written out as the `Value` portion of the HTML `OPTION` tag and only the `Name` portion will be displayed. IntraWeb does not use the `Value` tag and has no use in a normal IntraWeb application as this can be accomplished easier using Delphi code in the IntraWeb application. This option is useful if you are using `ScriptEvents`.

The `TIWComboBox.NoSelectionText String` property specifies the "no-selection" text of HTML listboxes and comboboxes (that must have a selected item).

The `TIWComboBox.RequireSelection Boolean` property specifies if a -- No Selection -- will appear. HTML does not allow for the user to unselect a combo or listbox otherwise.

The `TIWComboBox.ScriptEvents TIWScriptEvents` property can be used to add Javascript to control events to be processed on the client side. Scripts must be a Boolean evaluation or a function that returns a Boolean. If it evaluates to true, IntraWeb will continue processing the event itself if it uses the particular event. If it returns false, IntraWeb will not process the event further.

The `TIWComboBox.Text String` property can be used to modify the text that the control is displaying.

The `TIWComboBox.UseSize Boolean` property specifies whether to use the sizes defined at design-time or use the size assigned on generation of the HTML.

The `TIWComboBox` has an `OnChange` event handler that will fire some server-side code when the user changes the selection in the combobox. Warning: don't use this event handler too often, or your performance will suffer (since each event will trigger a response from the server), but use the `OnAsync` events instead (covered in the AJAX section).

TIWEdit

The TIWEdit component displays an editbox that can be used for user input, using the Text as main property.

The TIWEdit.PasswordPrompt Boolean property specifies whether or not this field will be used for password masked entry. If the value is set to True, the field displays a * (depending on the browser) instead of the actual text the user is typing.

The TIWEdit.Text String property contains the value of the TIWEdit box.

The TIWEdit.BGColor TIWColor property specifies the background colour for the editbox.

The TIWEdit.DoSubmitValidation Boolean property specifies whether or not validation is performed by other controls when the control is submitted.

The TIWEdit.Editable property specifies whether the TIWEdit should respond to user actions or not. If set to True, the TIWEdit will render as an input box in HTML, but if set to False, the TIWEdit will render as a text label.

The TIWEdit.FocusColor TIWColor property specifies the colour the TIWEdit gets when it receives the focus.

The TIWEdit.FriendlyName String property is used when displaying messages to the user about validation problems (for example with the Required property set to True).

The TIWEdit.MaxLength Integer property specifies the maximum length of text specified as a character count that the user will be permitted to enter.

The TIWEdit.ReadOnly Boolean sets the TIWEdit control read-only in the browser. See also the TIWEdit.Editable property.

The TIWEdit.Required Boolean property specifies whether or not user is required to enter a value in this field before they can submit the form.

The TIWEdit.ScriptEvents TIWScriptEvents property can be used to add Javascript to control events to be processed on the client side.

The TIWEdit has an OnSubmit event handler that is executed when a button is clicked to submit the value of the edit control to the server. This can be used for validation, for example.

TIWFile

The TIWFile component represents a file that can be uploaded from the client to the IntraWeb application. Inside the browser, it will display an editbox as well as a browse button next to it.

The TIWFile.ContentType String read-only property specifies the HTML content type of the file to be transferred.

The TIWFile.FileName String property contains the name of the transferred file.

The TIWFile.SaveToFile(AFilename: string) method can be used to save the file to a desired location on the web server machine where the IntraWeb application is running. The parameter AFilename should contain the name under which the file should be saved. The file name must be specified with full path. If this parameter is omitted, the file is saved under his default file name in the current directory.

The TIWFile.SaveToStream(AStream: TStream) method can be used to save the file on the IntraWeb server into a stream. The parameter AStream designates the stream on which the file should be saved. The stream must be already created.

The TIWFile has two event handlers: OnReceivedFile (which has AFileName: String as argument), and OnSubmit which is executed when a button is clicked to submit the value of the file control to the server.

TIWFlash

The TIWFlash component enables you to include Flash Files with your IntraWeb application. Like the TIWApplet component is this a nice way to integrate an external source into your IntraWeb application.

TIWHRule

The `TIWHRule` component displays a static horizontal rule, just like a `<hr>` in HTML. This is probably the simplest of all IntraWeb components, and has no events to respond to.

TIWImage

The `TIWImage` component can be used to display an image in the browser. You can load any image format (in the `Picture` property), and it will be converted to JPG when it's rendered, using the `JPegOptions` subproperties.

The `TIWImage.Picture` property specifies the picture to display. This picture can be dynamic and changed at run time.

The `TIWImage` has two events: an `OnClick` and an `OnMouseDown` event handler. Both can be used to create "hot" images and image maps.

TIWImageFile

The `TIWImageFile` component can also be used to display an image, but this time you don't load the image in the `Picture` property, but rather refer to the image with the `ImageFile` property (which can point to a local `Filename` or a full URL). Since there is no conversion involved, this is a more efficient way to use images in your IntraWeb applications compared to using the `TIWImage` component.

The `TIWImageFile.ImageFile` `TIWFileReference` property specifies the file to display. The `Filename` contains a path and filename to the image file.

Note: while the property editor allows you to select any file on the local computer, for the image to display properly in the users browser at run time the file must reside in the "Files" subdirectory of the application directory. If you do not have a "Files" sub directory, you must create one in your application directory.

Intraweb contains support for JPG files. To support GIF files, GIF support such as `GIFImage` must be installed into Delphi. The GIF support does not need to be compiled into the program at runtime, but must be loaded into Delphi so that `TIWImageFile` can properly display the GIF at design time.

The `TIWImageFile` has two events: an `OnClick` and an `OnMouseDown` event handler. Both can be used to create "hot" images and image maps (just like the `TIWImage` component).

TIWImageButton

The `TIWImageButton` is a `TIWImage` component that not only response to a `Click`, but also has both an `ImageFile` and a `HotImageFile` property, both with the usual `Filename` and `URL` property. The `HotImageFile` is shown when the mouse moves over the button.

The `TIWImageButton` has two events: an `OnClick` and an `OnMouseDown` event handler.

TIWList

The `TIWList` component can be used to produce a static list of items, just like the `` and `` HTML tags with `` for individual list items.

The `TIWList.Items` `TStrings` property holds the list of items to be displayed in the list. The `TIWList.Numbered` Boolean property specifies whether the list will be rendered as a numbered list (equivalent to the `` tag) or as an unnumbered list (equivalent to the `` tag).

TIWLabel

The `TIWLabel` component can be used to display a label of text. Useful in combination with a `TIWEdit` or `TIWDBEdit` component for example.

The `TIWLabel.AutoSize` Boolean property can be set to `True` to enforce the label to dynamically resize itself in order to fit the text in the `Caption` property.

The `TIWLabel.Caption` `TCaption` String property is the text which is displayed to the user. You can use the font to change how the text is displayed to the user.

The `TIWLabel.RawText` Boolean property can be set to True to make sure special characters will not be translated into HTML escape characters but emitted in raw unchanged format.

We'll examine the implementation of the `TIWLabel` component - as well as derived components - in the section on IntraWeb custom components, later in this courseware manual

TIWListbox

The `TIWListbox` component displays a listbox, with string items that we can add to the `Items` property.

The `TIWListbox.ItemIndex` Integer property specifies the currently selected item. A value of -1 specifies that no item is currently selected.

The `TIWListbox.Items` `TStrings` property contains the string items in the `TIWListbox`, which can be sorted or unsorted.

The `TIWListbox.MultiSelect` Boolean property specifies whether or not a multiple selection of items can be made. `MultiSelect` and `RequireSelection` are mutually exclusive.

The `TIWListbox.Sorted` Boolean property specifies the sorting of items. Setting `Sorted` to True causes the items to be sorted alphabetically. If the property is set to False, the items are presented as they are provided in the `Items` property.

The `TIWListbox.DoSubmitValidation` Boolean property specifies whether or not validation is performed by other controls when the control is submitted.

The `TIWListbox.Editable` property specifies whether the `TIWComboBox` should respond to user actions or not.

The `TIWListbox.ItemsHaveValues` Boolean property can be set to True if the items are of the format: `Name=Value`. The `Value` will be written out as the `Value` portion of the HTML `OPTION` tag and only the `Name` portion will be displayed. IntraWeb does not use the `Value` tag and has no use in a normal IntraWeb application as this can be accomplished easier using Delphi code in the IntraWeb application. This option is useful if you are using `ScriptEvents`.

The `TIWListbox.NoSelectionText` String property specifies the "no-selection" text of HTML listboxes and comboboxes (that must have a selected item).

The `TIWListbox.RequireSelection` Boolean property specifies if a `-- No Selection --` will appear. HTML does not allow for the user to unselect a combo or listbox otherwise.

The `TIWListbox.ScriptEvents` `TIWScriptEvents` property can be used to add Javascript to control events to be processed on the client side. Scripts must be a Boolean evaluation or a function that returns a Boolean. If it evaluates to true, IntraWeb will continue processing the event itself if it uses the particular event. If it returns false, IntraWeb will not process the event further.

The `TIWListbox.Text` String property can be used to modify the text that the control is displaying.

The `TIWListbox.UseSize` Boolean property specifies whether to use the sizes defined at design-time or use the size assigned on generation of the HTML.

The `TIWListbox` has an `OnChange` event handler that will fire some server-side code when the user changes the selection in the listbox. Warning: don't use this event handler too often (but use async events, which will be covered in the AJAX section).

TIWLink

The `TIWLink` component can be used to display a hyperlink in the browser that the user can click on.

The `TIWLink.Caption` `TCaption` String property specifies the text that is displayed in the link.

The `TIWLink.Color` `TIWColor` property specifies the background colour for the link.

The `TIWLink.ConfirmationString` property can be used to display a prompt to the user to confirm the `OnClick` action of the link when the user clicks on it.

The `TIWLink.DoSubmitValidation` Boolean property specifies whether or not validation is performed by other controls when the control is submitted.

The `TIWLink.ScriptEvents` `TIWScriptEvents` property can be used to add Javascript to control events to be processed on the client side. Scripts must be a Boolean evaluation or a function that returns a Boolean. If it evaluates to true, IntraWeb will continue processing the event itself if it uses the particular event. If it returns false, IntraWeb will not process the event further.

The `TIWLink.RawText` boolean property, by default set to false, can be used to control how the Caption of the `TIWLink` component has to be rendered as HTML. If you want to include your own HTML tags, then set the `RawText` property to true.

The `TIWLink` has an `OnClick` event handler that will execute some server-side code when the user indeed clicks on it.

TIWMemo

The `TIWMemo` component can be used to show a multi-line edit control, where the user can enter more than one line of text.

The `TIWMemo.Lines` `TStrings` property holds the contents of the multi-line edit control. Unlike the `TIWEdit`, however, the `TIWMemo` has no special event handlers.

TIWMenu

The `TIWMenu` component can be used to add a menu to your IntraWeb Page Form. Note that for the actual menu, it needs a `TMainMenu` component, which then needs to be assigned to the `AttachedMenu` property. Note that you still need to place the `TIWMenu` on the top of your IntraWeb Page Form (by default, it will be displayed where you placed the component, and not in the upper-left corner).

As a consequence of using a `TMainMenu` to delegate the menu itself, the `TMainMenu` menu items (events) are actually the ones that are executed, and the `TIWMenu` is only responsible for rendering and displaying the menu in a browser.

The `TIWMenu.AttachedMenu` `TMainMenu` property can be used to copy the items from a standard `TMenu` control (like `TMainMenu`) to the `TIWMenu`.

The `TIWMenu.AutoSize` `TIWMenuAutoSize` property specifies whether the control should be automatically sized to the browser's window or frame dimensions or not. Possible values are: `mnaNone` (menu is drawn according to `Width` and `Height` values), `mnaFullwidth` (menu is expanded to the full width of the window), or `mnaFullHeight` (menu is expanded to the full height of the window).

The `TIWMenu.ItemSpacing` `TIWSpaceItem` property specifies the spacing of the menu items. Possible values are: `itsNone` (items are not spaced at all) or `itsEvenlySpaced` (items are evenly spaced).

The `TIWMenu.MainMenuStyle` `TIWMenuStyle` property specifies various attributes, like font and colour, for the menu control.

The `TIWMenu.TextOffset` Integer property specifies the offset in pixels from the border of the menu control where the text will be drawn.

The `TIWMenu.TimeOut` Integer property specifies the interval of time after which the dropped down section of a menu closes automatically.

TIWProgressBar

The `TIWProgressBar` component can be used to display a progressbar. The progressbar is always shown horizontal only, and filled from left to right, it can display the progress value itself as well and use a colour.

The `TIWProgressBar.BGColor` `TIWColor` property specifies the background colour of the progressbar.

The `TIWProgressBar.Color` `TIWColor` property specifies the colour that is used for the progress portion of the bar (from left to right).

The `TIWProgressBar.Percent` `Integer` property specifies the progress value in percent from 0 to 100.

The `TIWProgressBar.ShowText` `Boolean` property specifies whether or not a text representation of the percentage will be displayed inside the progressbar.

TIWRadioGroup

The `TIWRadioGroup` component displays a group of radio button component, which can be filled by using the `Items` property.

The `TIWRadioGroup.ItemIndex` `Integer` property specifies the currently selected item. A value of -1 specifies that no item is currently selected.

The `TIWRadioGroup.Items` `TStrings` property contains the items in the `TIWRadioGroup`.

The `TIWRadioGroup.Layout` `TIWRadioGroupLayout` property specifies if radio buttons will be arranged vertically in a column or horizontally in a row.

The `TIWRadioGroup.Confirmation` `String` property can be used to display a prompt to the user to confirm the `OnClick` action of the link when the user clicks on it.

The `TIWRadioGroup` has no special event handlers when a radio button is clicked, but it does support the `OnClick` event handler.

TIWRectangle

The `TIWRectangle` component is like a big static panel (with its own colour) on which we can put other controls.

You can add text to the rectangle using the `Text` property. You can change the alignment of the rectangle by setting the `Alignment` and `VAlign` properties. You can change the appearance of the rectangle's border by modifying the `BorderOptions` property.

TIWRegion

The `TIWRegion` component can be used as a container for other IntraWeb controls, like the regular `TPanel` component of Delphi. All controls on a `TIWRegion` have the same parent, and the same relative position, so you can easily work with and position regions.

TIWText

The `TIWText` component can be used to display multiple lines of text (in the `Lines` property) without the ability to have any user input. This is an ideal way to include some "raw" HTML.

Note that four other `TIWxxx` components also support a `RawText` property to render them as raw HTML (the `TIWLabel`, `TIWLink`, `TIWURL`,)

TIWTimer

The `TIWTimer` component is a handy component: it fires the `OnTimer` event handler after every specified `Interval`. The event is fired at the client side in the browser (using JavaScript), but you must take care not to invoke `OnTimer` events while a previous one is still executing, because each `OnTimer` event will trigger a request to the web server application!

The `TIWTimer.Enabled` `Boolean` property specifies whether the timer is active or not. A disabled timer (`Enabled = false`) will not trigger any `OnTimer` event.

The `TIWTimer.Interval` `Integer` property specifies the interval of time in milliseconds after which the timer will call the action defined in the `OnTimer` event.

The `TIWTimer` has one obvious event handler: `OnTimer`, which is executed when the specified `Interval` is over (`Interval` is in milliseconds, so a value of 1000 is one second).

TIWGrid

The TIWGrid component displays a grid that we can fill with different information in each cell. In order to work with the TIWGrid, we have to set the ColumnCount and RowCount properties, as well as the Cell property (of type Array of Array of String, starting the index with row and then column).

The TIWGrid has two special event handlers: OnRenderCell (which is part of the TIWCustomGrid in case you want to create your own custom TIWGrid component) and OnCellClick which is fired when the user clicks inside a Grid Cell with the Clickable property set to True.

Warning: you don't want to call these events too often (for large Grids, this means that for every cell render or after each cell click, a server side event handler is fired).

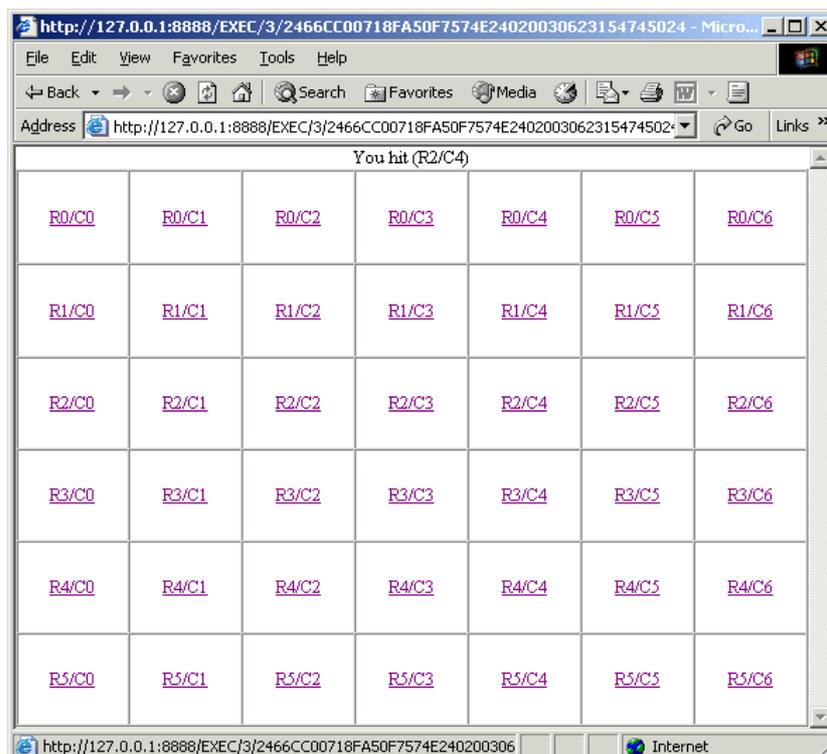
As an example, set ColumnCount to 7 and RowCount to 6 and write the following code for the FormCreate and IWGrid's OnCellClick event handler:

```

procedure TIWForm1.IWAppFormCreate(Sender: TObject);
var
  i,j: Integer;
begin
  for i:=0 to Pred(IWGrid1.RowCount) do
  begin
    for j:=0 to Pred(IWGrid1.ColumnCount) do
    begin
      IWGrid1.Cell[i,j].Text := Format('R%d/C%d',[i,j]);
      IWGrid1.Cell[i,j].Alignment := taCenter;
      IWGrid1.Cell[i,j].Clickable := True
    end
  end
end;

procedure TIWForm1.IWGrid1CellClick(const ARow, AColumn: Integer);
begin
  IWGrid1.Caption := Format('You hit (R%d/C%d)', [ARow, AColumn])
end;

```



TIWTreeView

The `TIWTreeView` component can be used to produce a static treeview or outline in the IntraWeb application. Note that there are no special events associated (when clicking on a node or opening/closing nodes).

Individual `TIWTreeView` nodes, however, can be assigned `OnClick` event handlers using the special `TIWTreeView` node editor.

TIWTreeViewItem

Drop a `TIWTreeView` component on an IntraWeb form, and click on the ellipsis for the `TreeNodeEx` property to start the `IWTreeView` items editor. If you right-click here, you'll find four possible actions: New Root Item, New Item, Delete Item and Edit Caption (although you can also edit the `Caption` using the Object Inspector). We have to start with a New Root Item (the treeview can contain more than one root), and then the subitems.

Each item can get five item images (for closed folder, document, minus, open folder, and plus), can be Expanded to show subnodes, and can respond to the `OnClick` event handler, which sends the itemnode itself as sender.

TIWURL

The `TIWURL` component can be used to show hyperlinks to external pages. A bit similar to the `TIWLink` component, with the difference that no `OnClick` event handler is present, and the `URL` property actually defines where you are going to after clicking on the component.

The `RawText` boolean property, by default set to false, can be used to control how the `Caption` of the `TIWURL` component has to be rendered as HTML. If you want to include your own HTML tags, then set the `RawText` property to true.

TIWURLWindow

The `TIWURLWindow` component is the equivalent of a DHTML `<IFRAME..>`. Use `TIWURLWindow` to create windows within the main window, but be aware that not all browsers support this.

The `URI` property can be given a value like <http://www.eBob42.com/courseware> or a reference to a local file.

TIWMPeg

The `TIWMPeg` component enables you to include MPeg movies with your IntraWeb application. I have no experience with this, yet.

TIWQuickTime

The `TIWQuickTime` component enables you to include QuickTime movies with your IntraWeb application. I have no experience with this, yet.

TIWCalendar

`TIWCalendar` is an IntraWeb class designed to render a calendar. The calendar can be customised, and each individual cell in the calendar can display user defined text. `TIWCalendar` works with localised strings. The day names and the display of the current date are made with values taken from the operating system.

Multiple IntraWeb Application Forms

Let's now examine how multiple IntraWeb forms can work together. Do *File | New - Other*, go to the VCL for the Web category in the Object Repository and double-click on the New Form to use the New Form Wizard to create a second IntraWeb Application Form. Where the first IntraWeb Application Form was named "IWForm1", the second one will be called "IWForm2" and saved in IWSecondForm.pas. In order to navigate from the first form (the main form) to the second one, we first need to add the second unit to the uses clause of the interface section of the first unit, and then need to drop a TIWButton component on each of the two IntraWeb Application Forms.

So, place a TIWButton component on the first IntraWeb Application Form, and set the Caption property to "Next Form".

In the OnClick event handler, write the following code:

```
procedure TIWForm1.IWButton1Click(Sender: TObject);
begin
  TIWForm2.Create(WebApplication).Show
end;
```

In this single line of code, we create the second IntraWeb Application Form, passing the WebApplication as owner (remember that WebApplication is a member field of all IntraWeb Forms), and directly calling the Show method to display the new form.

There are at least two things here that require special attention: *why use Show instead of ShowModal, and why not destroy the IntraWeb Application Form afterwards?* To answer the first question: there are no pop-up forms in web application, and hence a ShowModal is not needed. When a new IntraWeb Application Form is shown, it simply replaces the previous one. Or, more correctly, it is simply placed on top of the previous one, as if we were dealing with a huge stack of IntraWeb Application Form. And that's in fact exactly what's happening. And since we have a stack, there's no way the previous IntraWeb Application Form can wait and see what happens until the next form is destroyed. In fact, the call to Show is non-blocking (as we all know), so the previous IntraWeb Application Form is just waiting until it is activated again. And when an IntraWeb Application Form is no longer required, and should in fact be closed (in order to return to the previous Application Form), the programmer must call Release (and not Close or Free) to release the current IntraWeb Application Form and return to the previous one.

In order to implement this, drop a TIWButton component on the second form, put "Close" (or "Return") in its Caption property, and write the following single line of code in its OnClick event handler:

```
procedure TIWForm2.IWButton1Click(Sender: TObject);
begin
  Release
end;
```

That's it! Once we click on the first button, a new IntraWeb Application Form is created and shown, and if we click on the second button (or rather a button on the second IntraWeb Application Form), the IntraWeb Application Form is released again, and we return to the first IntraWeb Application Form.

Beware: this is one of those rare cases where you create something in one place, and release it somewhere else.

Final Release

Note that if you close your final form, then the application is terminated, and all you will see is a "200 OK" message in an otherwise empty browser window.

If you don't want that to happen, then you must make sure that the call to `Release` in the "main" IntraWeb Application Form is replaced by a call to `WebApplication.Terminate` with a goodbye message, as follows:

```
procedure TIWForm2.IWButton1Click(Sender: TObject);  
begin  
    Terminate('Thank you for using this IntraWeb application');  
end;
```

or a call to `WebApplication.TerminateAndRedirect`, sending the user to another URL with some more helpful information, as follows:

```
procedure TIWForm2.IWButton1Click(Sender: TObject);  
begin  
    TerminateAndRedirect(' http://www.eBob42.com ');  
end;
```

If you also pass a second string to `TerminateAndRedirect` (the message), then this message will be shown in the browser window for two seconds, before you will be redirected to the first string (the URL).

In all cases of using `Terminate` (or `TerminateAndRedirect`), the session will be destroyed.

Passing Information Around

The next chapter will cover IntraWeb State Management, but right now I want to show you that even without state management you can simply pass data from one IntraWeb form to another. Let's build a little Ping-Pong application, where we'll pass information from one form to another, using the current IntraWeb standalone application with two IntraWeb forms: `TIWForm1` and `TIWForm2`.

To implement this, place a `TIWEdit` on both `TIWForms`, and modify the `OnClick` event handler for the `TIWButton` on the `TIWForm1` as follows:

```
procedure TIWForm1.IWButton1Click(Sender: TObject);  
var  
    IWForm2: TIWForm2;  
begin  
    IWForm2 := TIWForm2.Create(WebApplication);  
    IWForm2.IWEdit1.Text := IWEdit1.Text; // copy data  
    IWForm2.Show  
end;
```

This will make sure that the contents of the `TIWEdit` from the `TIWForm1` is copied to the `TIWEdit` from the `TIWForm2`, thereby taking data with you from one place to another. And apart from IntraWeb controls, you can of course initialise any control or field from the `TIWForm2` from the `TIWForm1` event handler.

Getting Back

On `TIWForm2`, you can drop a `TIWButton` to release the second form again, which will remove the second form from the stack of forms, resulting in the first form being displayed again.

If you want to pass information back from `TIWForm2` to `TIWForm1`, then you must first store a reference to the first form (`TIWForm1`) in the second form, and initialise that field in the `IWForm1.IWButton1Click`.

This means that `TIWForm2` gets extended with a reference to `IWForm1` as follows (see next page):

```

type
  TIWForm2 = class(TIWAppForm)
    IWEdit1: TIWEdit;
    IWButton1: TIWButton;
    procedure IWButton1Click(Sender: TObject);
  public
    IWForm1: TIWForm1; // in Unit1.pas
  end;

```

With the following implementation of TIWForm2.IWButton1Click:

```

procedure TIWForm2.IWButton1Click(Sender: TObject);
begin
  if Assigned(IWForm1) and not IWForm1.Released then
  begin
    IWForm1.IWEdit1.Text := IWEdit1.Text;
    Release
  end
  else
    WebApplication.ShowMessage('Error: IWForm1 doesn't exist!', smAlert)
end;

```

Which also requires the following change to the TIWForm1.IWButton1Click:

```

procedure TIWForm1.IWButton1Click(Sender: TObject);
var
  IWForm2: TIWForm2;
begin
  IWForm2 := TIWForm2.Create(WebApplication);
  IWForm2.IWEdit1.Text := IWEdit1.Text; // copy data
  IWForm2.IWForm1 := Self; // copy self
  IWForm2.Show;
  //Release // try this to experience the error message of TIWForm2
end;

```

State Management

IntraWeb State Management is supported by the UserSession, where we can add any fields we want. Apart from using the IntraWeb Application Forms to pass information around, we can also use the UserSession to store information that belongs to our session (and never, ever use global variables because these are neither thread-safe nor unique for our specific request).

UserSession

So, let's now check out the UserSession unit that belongs to the IntraWeb application (if you checked the Create User Session option in the New IntraWeb Application wizard). By default, the UserSessionUnit contains the following source code (note that I fixed the comment, as it doesn't make much sense otherwise):

```

unit UserSessionUnit;
{
  This is a DataModule where you can add components or declare fields
  that are specific to ONE user. Instead of creating global variables,
  it is better to use this datamodule. You can then access it using the
  global UserSession variable.
}
interface
uses
  IWUserSessionBase, SysUtils, Classes;

```

```
type
  TIWUserSession = class(TIWUserSessionBase)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

implementation

{$R *.dfm}

end.
```

The UserSessionUnit contains a helpful comment that should tell developers not to use global variables, but to store these as fields in the TIWUserSession class.

Extending User Session

As an example, let's add a shopping cart to this TIWUserSession, in the form of a TStringList. During the execution of the IntraWeb application, anything can be placed on the shopping cart stringlist.

```
type
  TIWUserSession = class(TIWUserSessionBase)
  private
    { Private declarations }
  public
    { Public declarations }
    ShoppingCart: TStringList;
  end;
```

Note that we're already missing something important: creating the ShoppingCart. Right now, if we start using it, it will still be a nil pointer!

In order to create an instance of the TStringList when the User Session is created, we simply have to write some code in the OnCreate event handler of the TIWUserSession, as follows:

```
procedure TIWUserSession.IWUserSessionBaseCreate(Sender: TObject);
begin
  ShoppingCart := TStringList.Create;
end;
```

And if we ever need to clean up something from our session, we can do it in the OnDestroy event handler (which will be called when the session times out, or when the session is explicitly destroyed).

Using User Session

The next question is: how do we use the ShoppingCart from the UserSession? For this, we need to look back at the ServerController unit, which defines a global function UserSession that returns a TIWUserSession.

The implementation of this function is as follows:

```
function UserSession: TIWUserSession;
begin
  Result := TIWUserSession(WebApplication.Data);
end;
```

Where WebApplication is the instance of the IntraWeb application that belongs to our thread.

So inside our IntraWeb Application Forms, we can simply add the ServerController unit to our uses clause, and call the UserSession to return our specific TIWUserSession instance. And since we've added the ShoppingCart (or any of our custom user session fields) to the actual TIWUserSession class, they will also show up using Code Insight. No further casting needed.

So, inside any event of the IntraWeb Application Form, we can write code like the following:

```
procedure TIWForm1.IWAppFormCreate(Sender: TObject);  
begin  
    UserSession.ShoppingCart.Add('Delphi VCL Database Development')  
end;
```

To add the book Delphi VCL Database Development to our shopping cart. And this shopping cart in the User Session can be used at any time and any place in our IntraWeb application, as long as the session remains active.

Summary

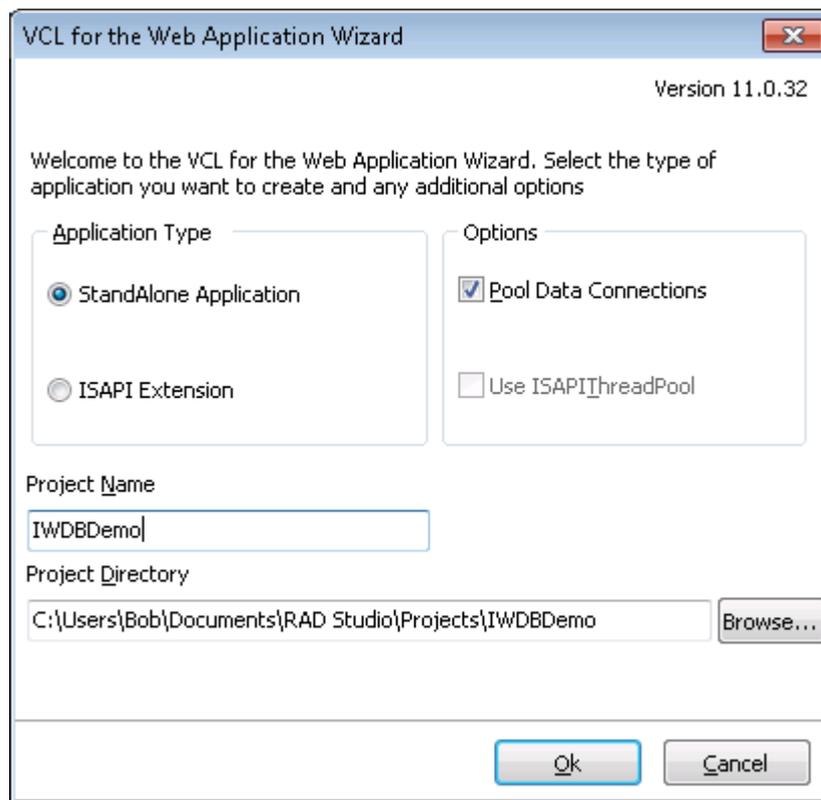
In this section, we have examined the IntraWeb Application, Server Controller, User Session and Form properties, events and methods, including how to pass information from one form to another, and using the User Session.

The next section will cover IntraWeb in combination with databases.

3. IntraWeb and Databases

IntraWeb supports databases and data access, and even allows us to share data modules and pool data modules (with with a little manual work, pool shared data modules).

In this section, we'll examine the data access part of IntraWeb. As example, we should create a new IntraWeb XI project, with the Pool Data Connections option checked (which we'll cover in detail in this section).



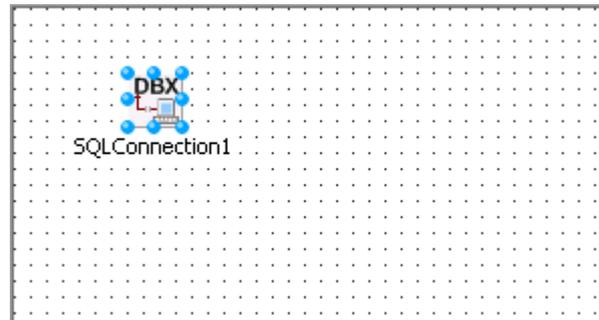
The resulting application IWDBDemo contains a DatamoduleUnit, a ServerController, a Unit1.pas (with the main form, so save that for example in IWMainForm.pas) and a UserSessionUnit.pas

Data Module

Let's turn our attention to the unit UserSessionUnit.pas, which contains the IWUserSession of type TIWUserSession, derived from TIWUserSessionBase which in turn is derived from TIWDataModule (which is an alias for TDataModule). As you can see, the TISUserSession is almost a regular data module (where the instance is only available for our session). As a consequence, we can place any data access, dataset and related components here. Since the Borland Database Engine (BDE) is "frozen" (and not recommended to use in web server applications anyway), and SQL Links even deprecated, this leaves dbGo for ADO, InterBase Express (IBX) and dbExpress (DBX4). Third party alternatives include Advantage Database Server (ADS) with its own TDataSet descendant component, which can also be used in combination with IntraWeb. In this case, I want to use dbExpress as database connectivity layer in web server applications. Using dbExpress, we can connect to InterBase, Oracle, DB2, Informix, MySQL, MS SQL Server and more using third-party drivers. If you know on beforehand that the database is limited to SQL Server or Access, you can use dbGo for ADO as well. Or you can use Advantage if that's the database of your choice, of course.

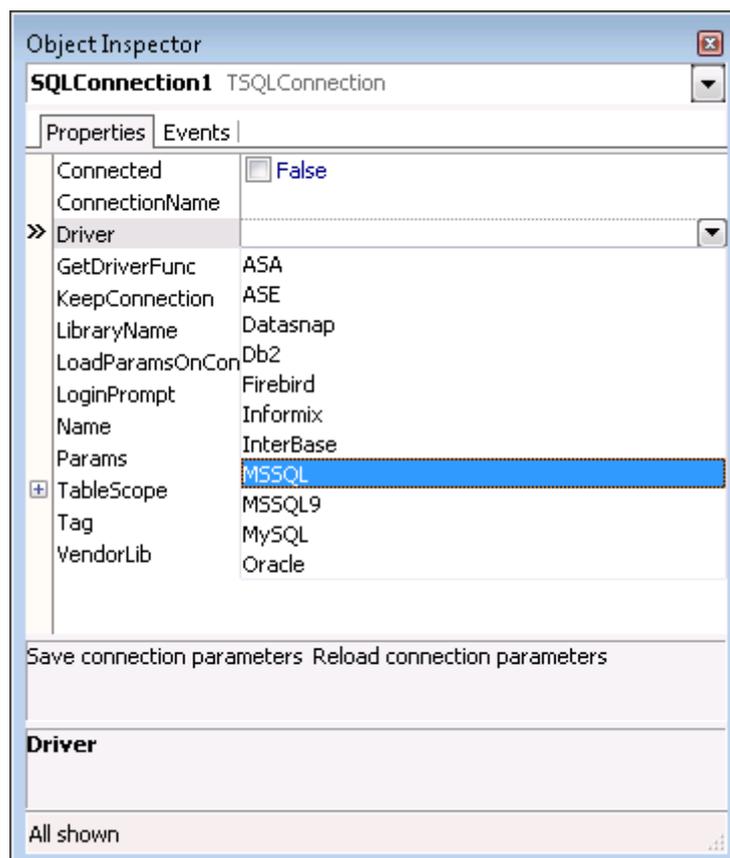
TSQLConnection

Using dbExpress, we need to start by placing a TSQLConnection component on the data module in the UserSessionUnit.



Using the Object Inspector, we can either assign a value to a predefined connection using the ConnectionName property, or configure the TSQLConnection component without depending on the connections definitions, by using the Driver property. Since this is easier when deploying the application (no need to deploy dbxConnections.ini), this example will only use the Driver property, and not the ConnectionName.

Open up the Driver property and select the database you want to use. The available database drivers depend on the edition of Delphi XE that you are using. Professional will show less drivers than Enterprise.



For the example in this section, select the MSSQL driver (or play along with a database connection of your own). Once MSSQL is selected, the Driver property gets a "+" prefix and can be opened to show the MSSQL specific subproperties. We need to specify the Database, HostName, UserName and Password subproperties. And also set the LoginPrompt to False, by the way.

Note that OS Authentication has to be set to False for SQL Server when deploying on a Windows server more recent than Windows 2000 (and an Internet Information Server version higher than 5).

For SQL Server we can use the example Northwind database for example, just like InterBase includes the EMPLOYEE.GDB database which can also be used of course.

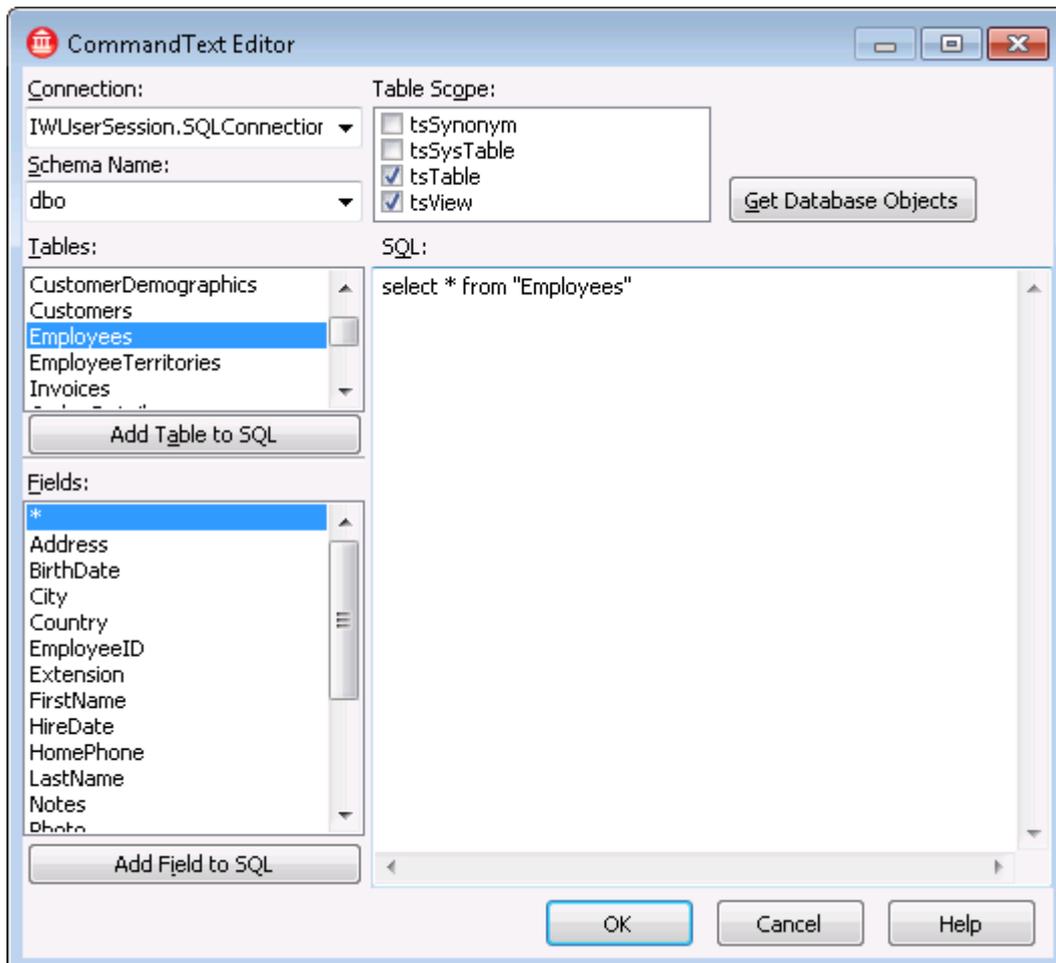
TSQLDataSet

For the examples in this section, I will use the Employees table from either of the SQL Server or InterBase databases.

To continue with the data module, place a first TSQLDataSet component (a hybrid component with almost TSQLTable, TSQLQuery and TSQLStoredProc functionality), and call it sqlEmployees. Point the SQLConnection property of the TSQLDataSet component to the TSQLConnection component.

Double-click on the CommandText property of the TSQLDataSet component to start the dbExpress Query CommandText Editor. Select "dbo" as Schema name (click on Get Database Objects to see the list of tables), and then use this dialog to build a simple SQL query, like

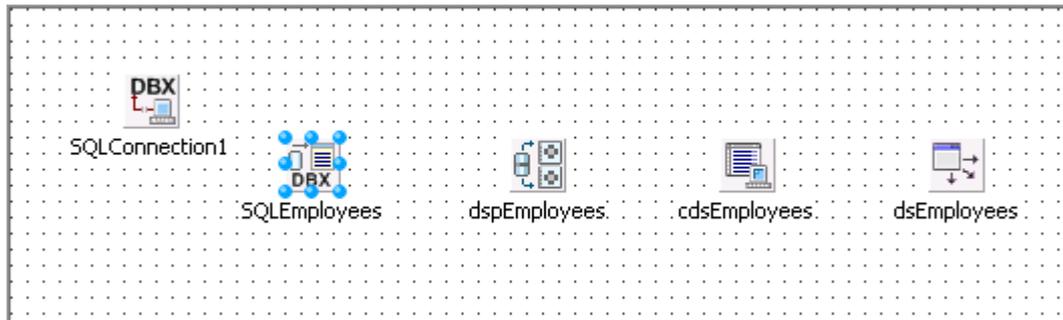
```
SELECT * FROM Employees
```



In order to be able to really use the data from the TSQLDataSet component, we need to place three additional components (from the Data Access category of the Delphi Tool Palette), namely a TDataSetProvider, a TClientDataSet and a TDataSource component. Place all three components on the data module.

Give the name property of the TDataSetProvider component the value dspEmployees, set the name property of the TClientDataSet component to cdsEmployees, and finally set the name of the TDataSource to dsEmployees. Point the DataSet property of the dspEmployees TDataSetProvider to the sqlEmployees TSQLDataSet component, point the ProviderName property of the cdsEmployees TClientDataSet component to the dspEmployees TDataSetProvider component, and finally point the DataSet property of the dsEmployees TDataSource component to the cdsEmployees TClientDataSet component. Set the dspEmployees's UpdateMode to upWhereChanged.

The data module in the UserSessionUnit should now roughly look as follows:



IW Data Controls

Now that we have the data module, it's time to move back to the IntraWeb forms. Starting with the "regular" TIWAppForm, we should look at the components of the IW Data category from the Delphi Tool Palette.

There are 14 data-aware controls in the IW Data category, namely: TIWDBCheckBox, TIWDBComboBox, TIWDBEdit, TIWDBGrid, TIWDBImage, TIWDBLabel, TIWDBListbox, TIWDBLookupListbox, TIWDBLookupCombobox, TIWDBFile, TIWDBMemo, TIWDBNavigator, TIWDBText, and TIWDBRadioGroup.

TIWDBCheckBox

The TIWDBCheckBox component is a data-aware TIWCheckBox component.

TIWDBComboBox

The TIWDBComboBox component is a data-aware TIWComboBox component.

TIWDBEdit

The TIWDBEdit component is a data-aware TIWEdit component.

TIWDBGrid

The TIWDBGrid component is a data-aware TIWCustomGrid component, which means it does not expose the OnCellClick event handler.

For a detailed overview of the capabilities of the TIWDBGrid component, see the continued demo section a few pages from now.

TIWDBImage

The TIWDBImage component is a data-aware TIWImage component.

TIWDBLabel

The TIWDBLabel component is a data-aware TIWLabel component.

TIWDBListbox

The TIWDBListbox component is a data-aware TIWListbox component.

TIWDBLookupListbox

The TIWDBLookupListbox component is a component that will display a listbox filled with lookup items that can be used to fill a field (when the lookup itself is coming from another dataset). Just like the regular Delphi TLookupListbox, using a DataSource and DataField, as well as ListSource and ListField (as well as KeyField).

The TIWDBLookupListbox has a special OnChange event handler.

TIWDBLookupCombobox

The TIWDBLookupCombobox component is a component that will display a combobox filled with lookup items that can be used to fill a field (when the lookup itself is coming from another dataset). Just like the regular Delphi TLookupCombobox, using a DataSource and DataField, as well as ListSource and ListField (as well as KeyField).

The TIWDBLookupCombobox has a special OnChange event handler.

TIWDBFile

The TIWDBFile component is a data-aware version of the TIWFile component, where a database stream field can be used to place the uploaded file.

TIWDBMemo

The TIWDBMemo component is a data-aware version of the TIWMemo component.

TIWDBNavigator

The TIWDBNavigator component is a database navigator in the IntraWeb Page Form.

The TIWDBNavigator.VisualButtons property specifies how many - and which - of the 10 possible buttons are displayed. This is useful if you want to display read-only actions only (no Edit, Post, Cancel, Insert and Delete for example).

The TIWDBNavigator.CustomImages property specifies custom images (either a Filename or URL) for each of the 10 buttons, both in enabled and in disabled ways.

The TIWDBNavigator.Confirmations property can be used to enter special confirmation messages for each of the 10 actions. Three message have been included already, for the Cancel, Delete and Post actions. You can translate these messages, or add other messages as well.

The TIWDBNavigator.Orientation property can switch between a orHorizontal or orVertical orientation (looks funny at first sight).

Each button on the TIWDBNavigator component has its own OnClick event handler. Or more specifically, the TIWDBNavigator has an OnCancel, OnDelete, OnEdit, OnFirst, OnInsert, OnLast, OnNext, OnPost, OnPrior and OnRefresh event handler.

These events are nice to write code for. However, you must realise that each of these events results in a message being posted to the server, and response being sent back to the client.

TIWDBText

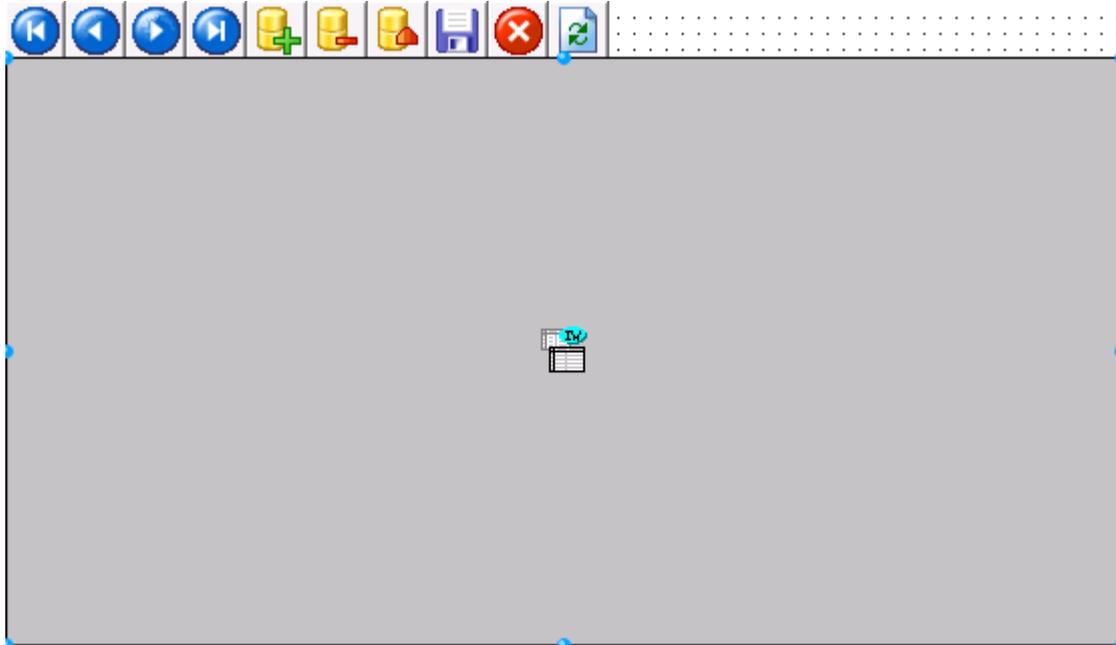
The TIWDBText component is just a data-aware version of the TIWText component, which can be used to display raw HTML from a database field (which is useful if you have a database table that actually contains string or memo fields that are filled with "raw" HTML that you want to display as it was meant to be).

TIWDBRadioGroup

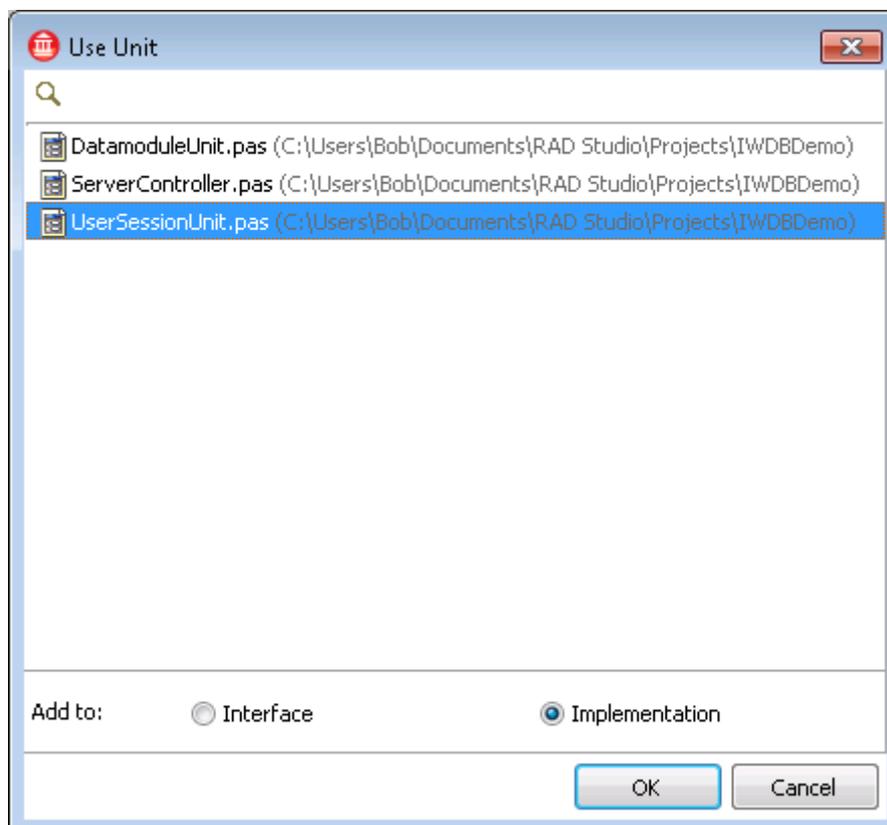
The TIWDBRadioGroup component is a data aware TIWRadioGroup control.

Continued Demo

Let's continue with our example and place IW Data controls on the IntraWeb Application Form. Start with a TIWDBNavigator and TIWDBGrid component. Set the Align property of the TIWDBNavigator to alTop, and finally set the Align property of the TIWDBGrid to alTop (or alClient if you want).



To make a connection with the data module, we need to add the UserSessionUnit to the uses clause (using Alt+F11).



After this step, we can connect the DataSource property of the TIWDBNavigator and the TIWDBGrid to IWUserSession->dsEmployees.



Note that we don't see any data at design-time, like we're used to in Delphi, not even if the datasets are made active. This is something to get used to, but remember that the main purpose of IntraWeb is to produce run-time HTML, not to provide a "live" design-time view as well (it would be nice, but is certainly not necessary in my view).

Also note that it's generally not a good idea to actually activate the datasets at design-time, since this may mean that the project gives long delays when you try to open it and the database is unavailable (for example if you open the project on a disconnected client). The best way is to activate the TSQLConnection when the session is started and the data module is created. In fact, if you activate the cdsEmployees then this will set the entire chain in motion, so the only thing we need is the following:

```
procedure TIWForm1.IWAppFormCreate(Sender: TObject);
begin
  UserSession.cdsEmployees.Active := True
end;
```

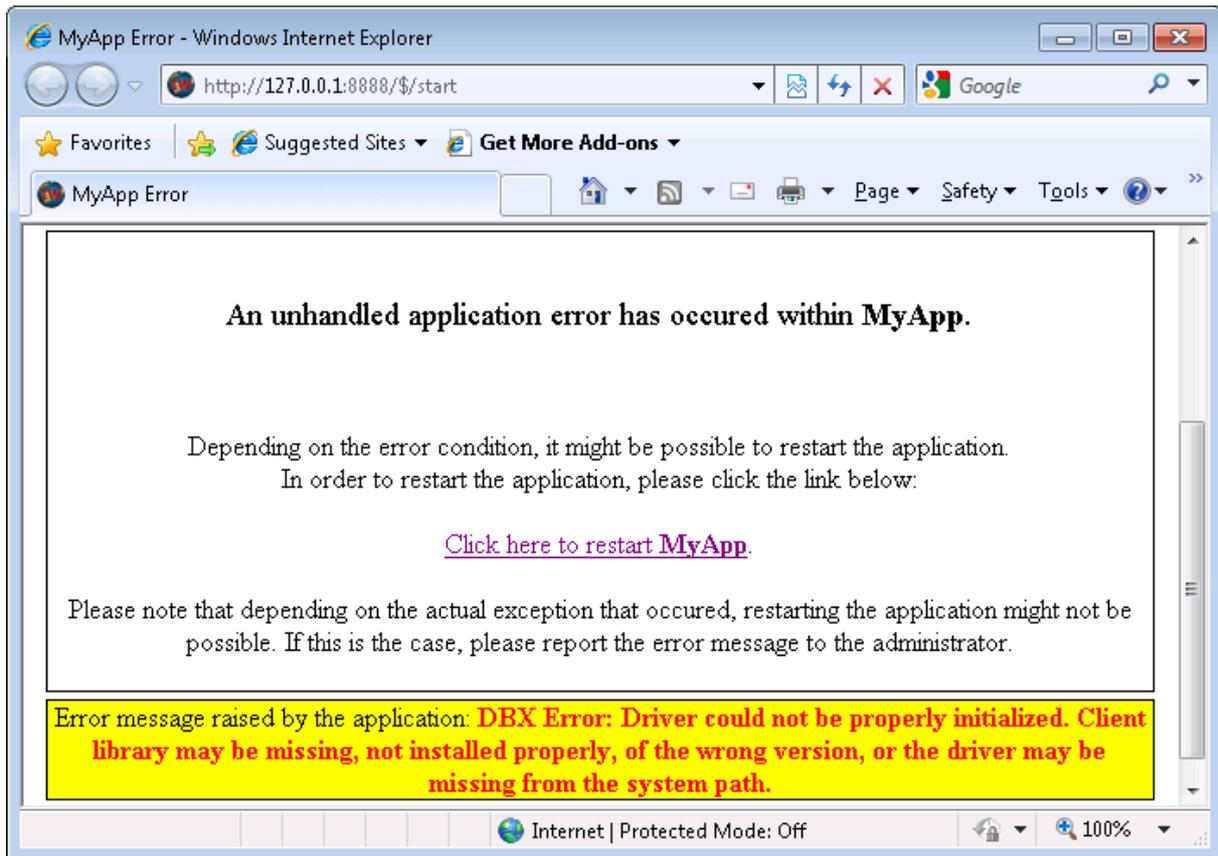
Note that we need to add the ServerController unit to the uses clause in order to be able to call the UserSession function.

Alternately, you can activate the cdsEmployees when the user session is created (which happens before the main form is created), as follows in the UserSessionUnit.pas:

```
procedure TIWUserSession.IWUserSessionBaseCreate(Sender: TObject);
begin
  cdsEmployees.Active := True;
end;
```

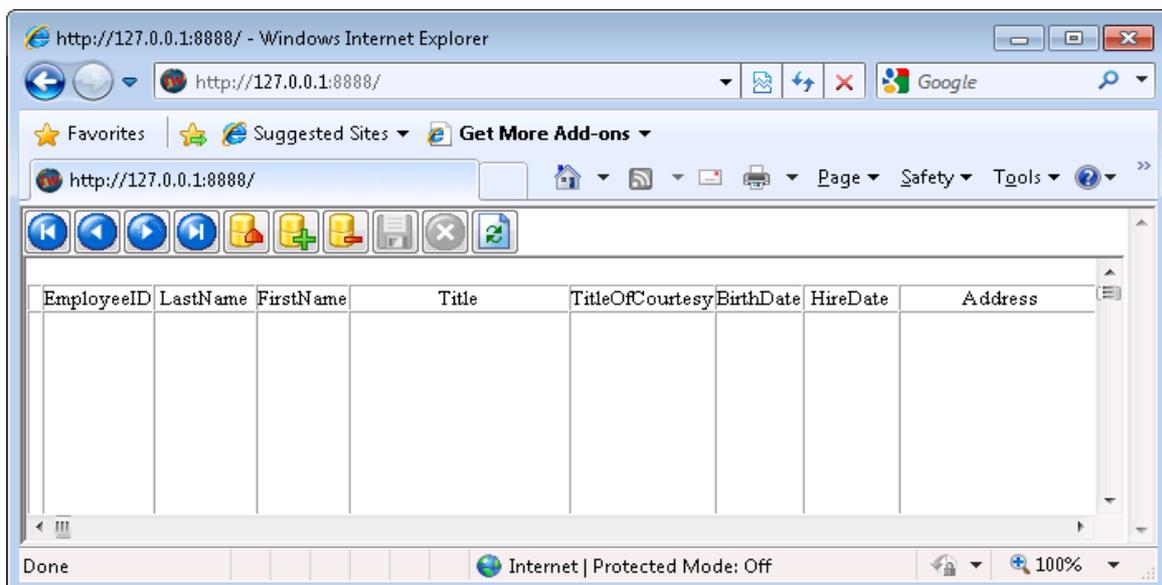
Go to the TIWDBGrid component, and set the dbIndicator subproperty of the Options property to True, so we can always see the current record in the grid.

Now, compile and run the application, and view <http://localhost:8888> in the browser to get the following result:



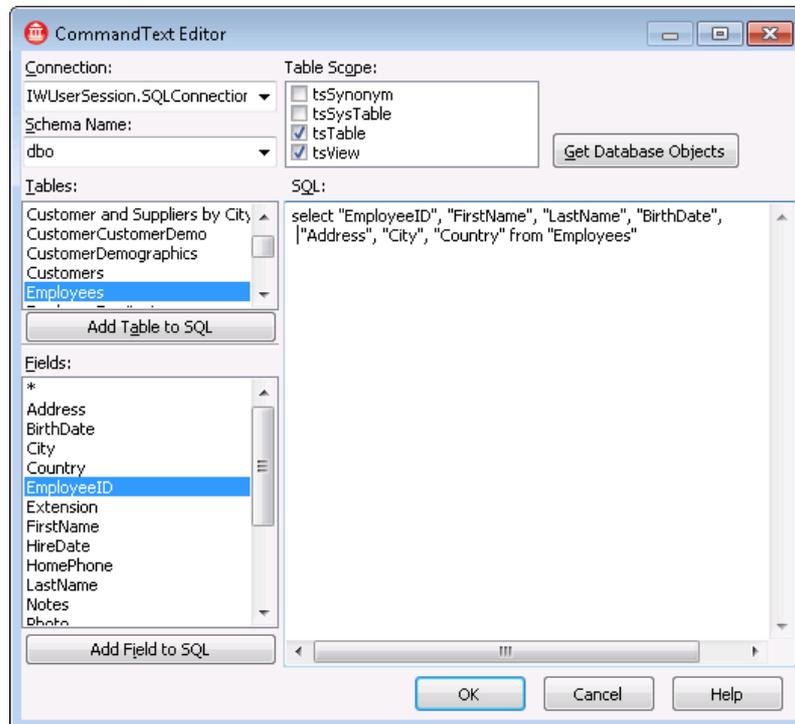
The problem may be caused by the fact that the DBX4 driver dbxmss.dll cannot be found by the IntraWeb application. We can find and copy the DBX4 driver dbxmss.dll from C:\Program Files\Embarcadero\RAD Studio\8.0\bin to the current application directory and restart it (the executable), before clicking on restart in the browser.

However, chances are that the problem will persist after copying the dbxmss.dll to your application directory. In that case, it's good to realise that dbExpress may require COM initialization when talking to SQL Server, so in the ServerController unit we should select the Server Controller and set the property ComInitialization to ciMultiThreaded. That will solve the error and present you with the following output:

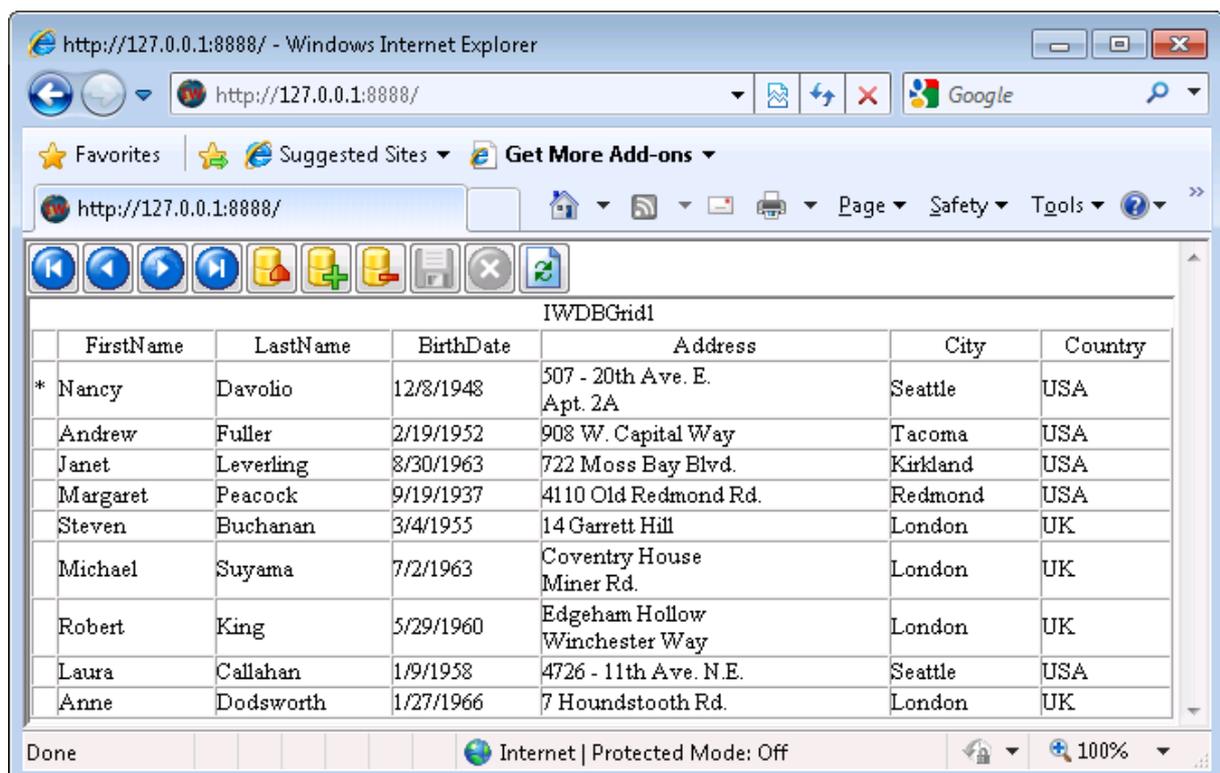


The problem with the layout (horizontal scrollbar!) is caused by the Photo field, which is not shown as image but as quite long binary field.

We should remove some fields from the TIWGrid. This can be done at several levels: in the TIWGrid, in the TClientDataSet (using the Fields Editor), in the TSQLDataSet (again in the Fields Editor) or in the original SQL. The best way is to eliminate unwanted fields as soon as possible, so we should change the SQL to a query selecting only a few fields with the following command:



This gives the following result:



TIWDBGrid Usage

By default, the TIWDBGrid is only suited to present data in a read-only way: there is no direct way to edit the data. There used to be an indirect way, which I will demonstrate, but which is no longer working.

But first, let's perform some general TIWDBGrid configuration steps. We can use the Caption property to set a title on top of the IWDBGrid like "Employees", and we can use the FooterRowCount property to add additional rows at the bottom of the grid (for summary information for example). After the data has been rendered into the cells the TIWDBGrid will begin rendering the footer rows by starting with negative row counts, counting from -FooterRowCount to -1 for the last row. So, set the FooterRowCount property to 1, and implement the OnRenderCell event handler as follows:

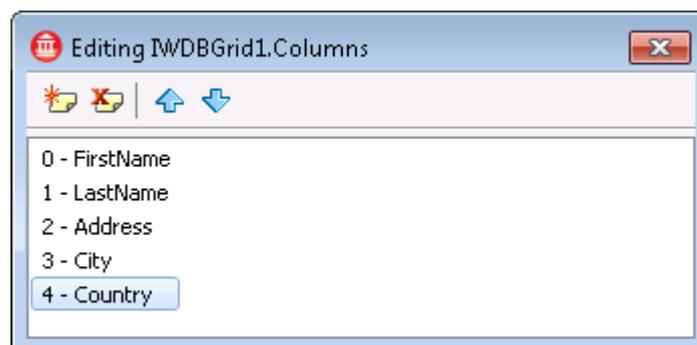
```
procedure TIWForm1.IWDBGrid1RenderCell(ACell: TIWGridCell;  
  const ARow, AColumn: Integer);  
begin  
  if ARow = -1 then // Footer  
    if AColumn = 0 then  
      ACell.Text := 'The footer of the TIWDBGrid'  
end;
```

We can specify the background colour of the grid with the BGColor property (for example to \$00FFFFCC). A nice effect can be obtained by setting the RowAlternateColor property to another value (for example \$00CCFFFF), in order to give alternate rows different colours. Also, the RowCurrentColor can be used to specify the colour of the current selected row (for example \$00CCCCFF).

Also, if we've set the RollOver property to True, then the row where the mouse is over will be shown with a certain colour indicated by the value of RollOverColor (for example \$00FFCCFF).

We can configure the way the TIWDBGrid refreshes itself with the RefreshMode property. Values can be rmAutomatic (data will be refreshed on each render request unless the dataset is in edit or insert mode), rmAlways (data will be refreshed on every render request. If the dataset is in edit or insert mode an exception will be raised), or rmManual (data will be refreshed only when the RefreshData method is explicitly called). Note that only the data inside the grid itself is refreshed, but never the rows of the dataset itself.

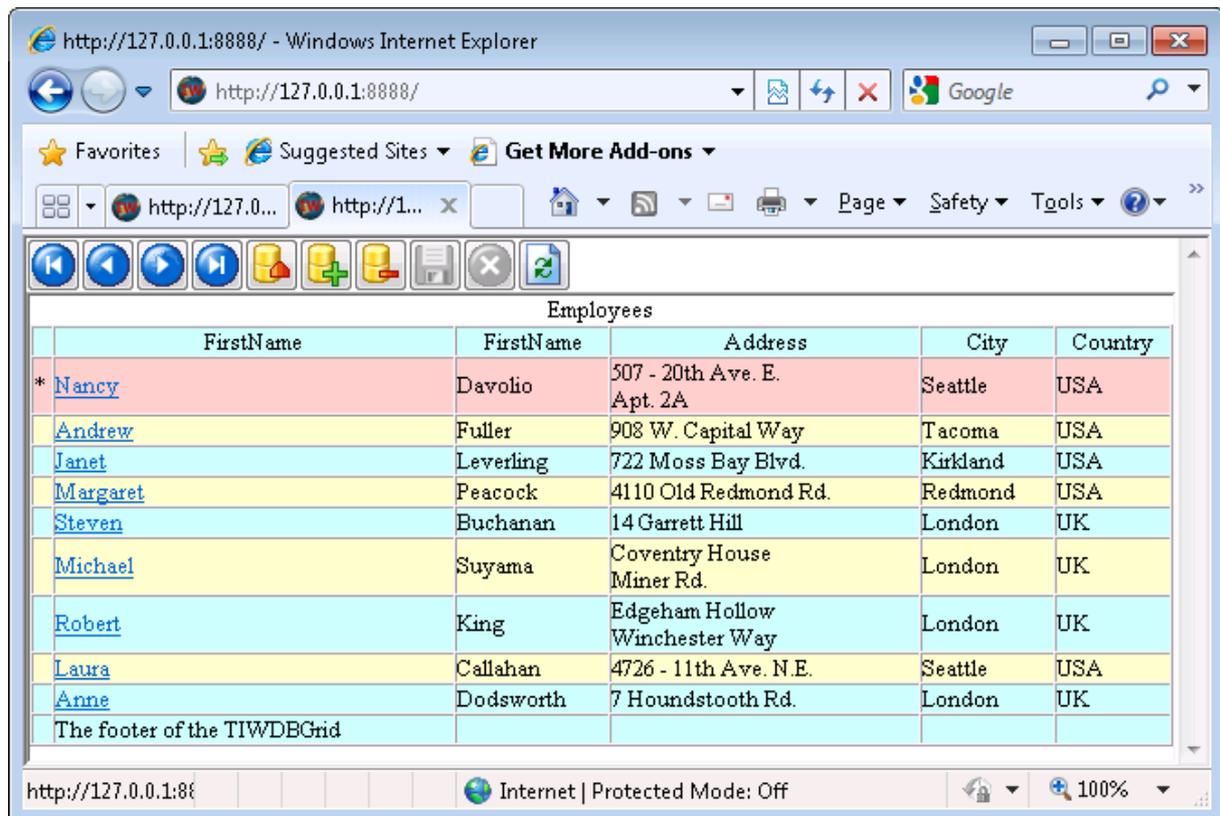
In order to configure the individual columns of the grid, we need to click on the Columns property of the TIWDBGrid component to start the IWDBGridColumn editor. Using this dialog, we can add explicit columns to the IWDBGrid (which by default will show all fields, but if you add one or more columns it will only show the persistent columns). For our example, I want to use five columns, so press the <insert> key five times. Select each of the TIWDBGridColumn components in turn, and connect their DataFields to the following fields: FirstName, LastName, Address, City and Country (skipping the EmployeeID and BirthDate fields).



Note that for a real-world example you may want or have to add all fields and connect them all to one of the fields of the Employees table, but I only want to show the techniques here, so I don't need them all.

Each of the individual column fields has a number of useful properties, such as the Visible property (in case you've changed your mind and decided not to show the EmployeeID field after all). Another useful property is the LinkField property, which can be used if you want to turn a field of the table (i.e. a column of the grid) to a set of clickable values. If you specify a value for the LinkField property, then the value (for example of the FirstName field) will turn into a clickable link, and when the user clicks on it, then the value of the LinkField is returned with the special click event. If you've just hidden the EmployeeID field, then it may be a good idea to use the Employees ID field as LinkField for the FirstName field for example (assuming Name is unique).

The result of this all, with the first record being the current record, and mouse cursor not over a record, can be seen below:



The only downside is that nothing happens if we click on the link for the FirstName column, because we still have to implement the OnClick event handler for the FirstName TIWDBGridColumn. This can be implemented - hardcoded - as follows:

```

procedure TIWForm1.IWDBGrid1Columns0Click(ASender: TObject;
  const AValue: String);
begin
  WebApplication.ShowMessage(AValue);
end;

```

Note that this only works for the FirstName column, since the other columns do not have a LinkField defined, and hence will not be turned into a hyperlink representation.

Apart from the OnClick event handler, the TIWDBGridColumn also have an OnTitleClick event handler. This one can be used to sort the table on that particular field for example (sharing the same code for all columns):

```

procedure TIWForm1.IWDBGrid1ColumnsTitleClick(Sender: TObject);
begin
    UserSession.cdsEmployees.IndexFieldNames :=
        TIWDBGridColumn(Sender).DataField
end;

```

It helps if you also specify in the footer on which field the grid is sorted:

```

procedure TIWForm1.IWDBGrid1RenderCell(ACell: TIWGridCell;
    const ARow, AColumn: Integer);
begin
    if ARow = -1 then
        if AColumn = 0 then
            if UserSession.cdsEmployees.IndexFieldNames <> '' then
                ACell.Text := 'Sorted by ' + UserSession.cdsEmployees.IndexFieldNames
end;

```

The result is a grid that can be sorted by column (like City), uses alternating colours, as well as a different colour to indicate the current or mouseover (third row) rows:

The screenshot shows a web browser window displaying a table titled "Employees". The table has six columns: "FirstName", "FirstName", "Address", "City", and "Country". The rows are color-coded in an alternating pattern. The third row (Nancy Davolio) is highlighted in pink, indicating it is the current row. The footer of the table reads "Sorted by City".

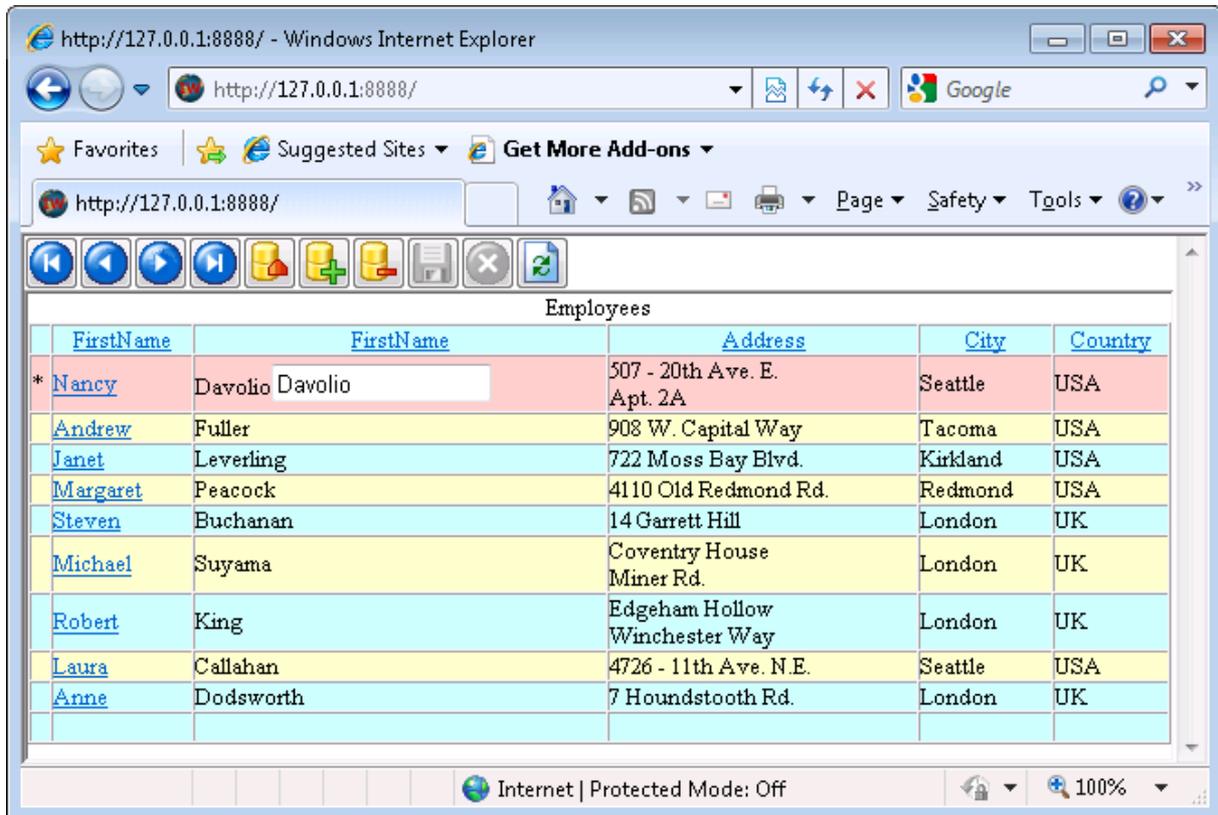
	FirstName	FirstName	Address	City	Country
	Janet	Leverling	722 Moss Bay Blvd.	Kirkland	USA
	Steven	Buchanan	14 Garrett Hill	London	UK
	Michael	Suyama	Coventry House Miner Rd.	London	UK
	Robert	King	Edgeham Hollow Winchester Way	London	UK
	Anne	Dodsworth	7 Houndstooth Rd.	London	UK
	Margaret	Peacock	4110 Old Redmond Rd.	Redmond	USA
*	Nancy	Davolio	507 - 20th Ave. E. Apt. 2A	Seattle	USA
	Laura	Callahan	4726 - 11th Ave. N.E.	Seattle	USA
	Andrew	Fuller	908 W. Capital Way	Tacoma	USA
	Sorted by City				

There's one more thing left to do: turn this grid into a way to edit the fields as well, for example the Price (which is still empty at this time). For this, we again need to look at the TIWDBGridColumn, and specifically to the Control property.

In case of the LastName, I want to edit it, so we need a TIWDBEdit component. Just place it somewhere on the form, since it will be "controlled" by the grid. Make sure to point the DataSource property to the IWUserSession->cdsEmployees, and the DataField to LastName.

Now, open up the Columns property editor from the TIWDBGrid again, select the Price column, and point its Control property to the TIWDBEdit.

This will result in the editbox being displayed next to the (original) value in the column. The user can now make changes to the field value, and move to another record or explicitly post the value to the ClientDataSet.



Note by the way that we are using a dbExpress dataset (via the TDataSetProvider and TClientDataSet), so we have to make sure that all Posts done to the TClientDataSet are only done in memory, and still have to be applied to the underlying database using the ApplyUpdates method of the TClientDataSet. This is needed in both the OnAfterPost and the OnAfterDelete event handlers of the TClientDataSet, since these are the two methods that can make changes:

```

procedure TIWUserSession.cdsEmployeesAfterPostOrDelete(DataSet: TDataSet);
begin
  (DataSet as TClientDataSet).ApplyUpdates(0)
end;

```

The result is a TIWDBGrid where we can edit the Price field of the current record.

If you don't like the fact that we still see the original value next to the TIWDBEdit box, then you can write some additional code in the OnRenderCell event handler, for example as follows (note the use of the RowIsCurrent method that I can call to determine if we've in the current row of the grid):

```

procedure TIWForm1.IWDBGrid1RenderCell(ACell: TIWGridCell;
  const ARow, AColumn: Integer);
begin
  if ARow = -1 then // Footer
  begin
    if AColumn = 0 then
      if UserSession.cdsEmployees.IndexFieldNames <> '' then
        ACell.Text := 'Sorted by ' + UserSession.cdsEmployees.IndexFieldNames
  end
end

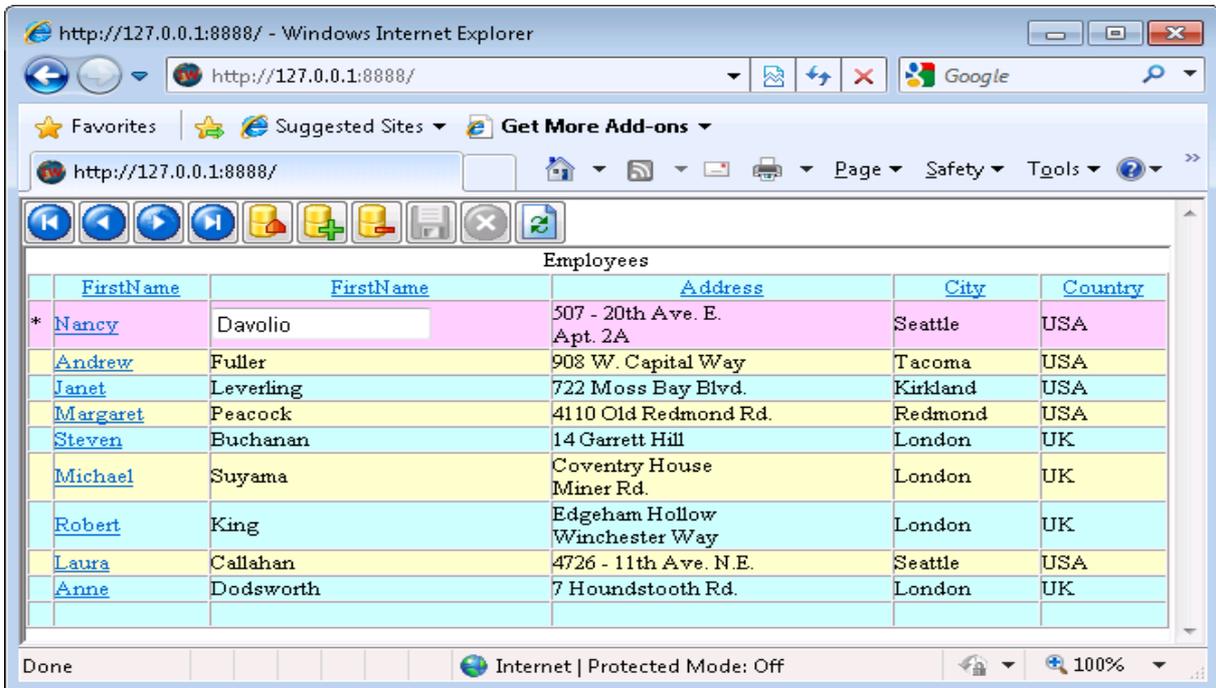
```

```

else
  if IWDBGrid1.RowIsCurrent then // current row ??
    if AColumn = 1 then ACell.Text := '' // clear, only show Control
end;

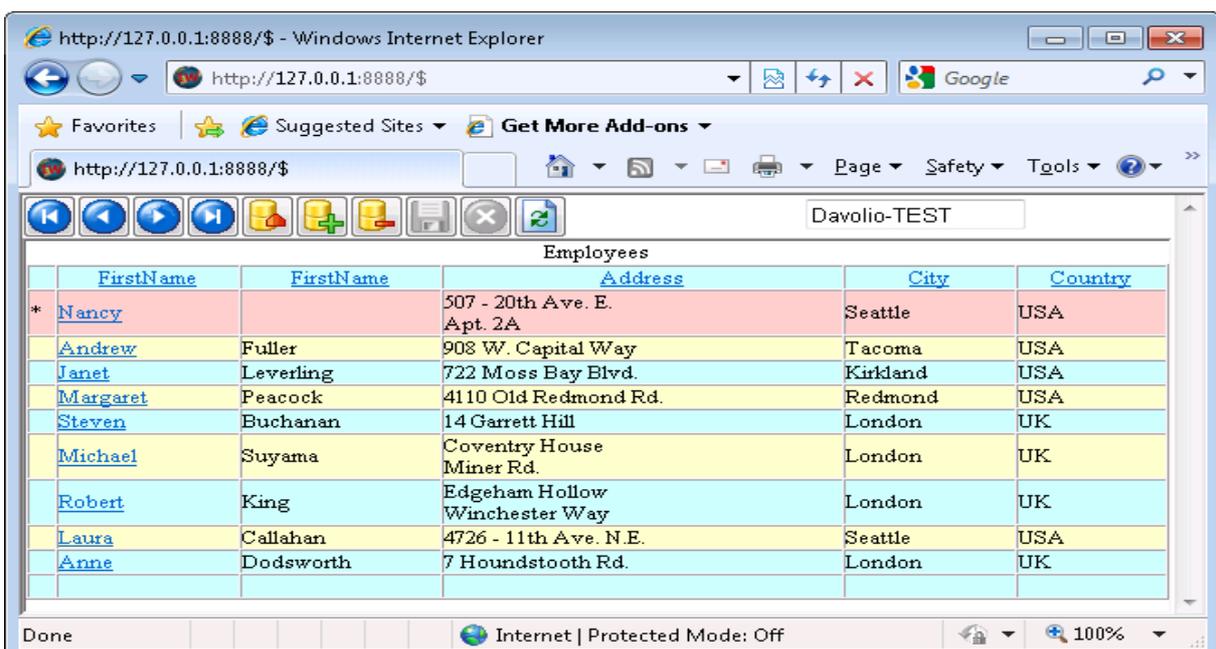
```

This will make sure that the column only shows the TIWDBEdit, and no text on the left side of it.



Unfortunately, editing values in the IWDBGrid will have no effect when posting the modified values (this used to work fine with previous versions). Using the IWDBEdit controls without the IWDBGrid works just fine, fortunately.

The only step needed to make the example work – with a stand-alone TIWButton – is to remove the TIWDBEdit button from the Control reference of the Price column, and let the TIWDBEdit control remain visible on the Application Form.



Sharing VCL data modules with VCL for the Web

One of the big advantages of VCL for the Web (aka IntraWeb) is the fact that you can reuse existing VCL data modules, as part of the UserSessionUnit.

The trick is as follows: in your project group that contains the VCL application (with the data module), create a new VCL for the Web application, and make sure to check the "Create User Session" option, so a UserSessionUnit.pas file is created with your project as well.

When the new VCL for the Web project is created, right-click on the project target and select Add... to add a new file to the project. In the dialog that follows, select the VCL data module from your existing VCL project.

Then, check the main project file and make sure the code to automatically create the data module is removed (since we cannot have just one data module, but we must have one for each incoming request):

```
//Application.CreateForm(TDataModule1, DataModule1);
```

This is the first step in sharing the single data module in both the VCL Win32 / .NET and the VCL for the Web application.

Once the VCL data module is added to your VCL for the Web project, open the UserSessionUnit.pas file. Press Alt+F11 to make sure the VCL data module is added to the uses clause of the UserSessionUnit.pas unit.

Note that inside the UserSessionUnit unit, the TIWUserSession class is already derived from a TIWUserSessionBase and represented as a data module.

However, this is *not* the data module you want to use, since that won't allow you to share your business logic between the VCL Win32 / .NET and VCL for the Web application.

Instead, we should add a field to the TIWUserSession for your data module, and implement a constructor of the TIWUserSession that will create the data module, as follows:

```
type
  TIWUserSession = class(TIWUserSessionBase)
  private
    { Private declarations }
  public
    { Public declarations }
    MyDataModule: DataMod.TDataModule1;
    constructor Create(AOwner: TComponent); override;
  end;
```

The implementation of the constructor is as follows:

```
constructor TIWUserSession.Create(AOwner: TComponent);
begin
  inherited;
  MyDataModule := DataMod.TDataModule1.Create(Self);
end;
```

This is the second step, but we're not done yet.

In our VCL for the Web Forms, we obviously want to connect to the DataModule at design-time, and don't want to write complex code to connect to the UserSession.MyDataModule.whatever datasets and fields.

For this, we usually use the global DataModule1 instance variable in the data module unit, declared as follows:

```
var
  DataModule1: TDataModule1;
```

However, that's not a good idea in this case, since there can be many instances of the data module (which is likely since the VCL for the Web application will receive many incoming requests, each having a User Session with a unique instance of the Data Module).

So using the global variable is a no-no here. Instead, we should change this to a function that will return the specific field from the UserSession, as follows:

```
{ $IFDEF IW }  
function DataModule1: TDataModule1;  
{ $ELSE }  
var  
    DataModule1: TDataModule1;  
{ $ENDIF }
```

And now we need to implement that function in the implementation section of the Data Module, as follows:

```
implementation  
  
{ $R *.dfm }  
  
{ $IFDEF IW }  
uses  
    ServerController, UserSessionUnit;  
  
function DataModule1: TDataModule1;  
begin  
    Result := UserSession.MyDataModule  
end;  
{ $ENDIF }
```

Now, we can add the Data Module unit to the uses clause of our IntraWeb Forms, and use the tables, queries, dataset and data sources in our IntraWeb code.

Just make sure to define IW when compiling the VCL for the Web edition of the data module, and remove that define when you compile the regular Win32 or .NET version.

The best way is to define IW in the "Conditional defines" box of the Project Options for the IntraWeb application, so it's only used in that case (beware that you still need to do a Build All if you switch from one project to another, since the Conditional define is not identified as a change in source code by the compiler; but that's another story).

Pool Data Connections

Another useful feature of VCL for the Web is the ability to pool data connections. This means that instead of creating instances of data modules for each user session, we maintain a pool of data modules, and each user can grab an instance from the pool when it needs one. This keeps the number of (open) connections to your database lower compared to a situation where every user session has one.

The good news is that IntraWeb offers this ability out-of-the-box, with the Pool Data Connections check box in the New VCL for the Web Application Wizard (see the screenshot on page 55 for example). The bad news is that it's not easy to add this ability later (if you don't know what to add). Which is why I will now create a new project, using the Pool Data Connections option enabled, and then explain what it consists of, how it works, and how we can change existing projects – that were created without the Pool Data Connections option enabled – to support the pooling of data connections as well.

If you create a new VCL for the Web project, using the Pool Data Connections as well as the Create User Session options, then the result is a project file, as well as a DatamoduleUnit, a ServerController unit, a main IntraWeb form unit and a UserSessionUnit.

DatamoduleUnit

The DatamoduleUnit is new compared to generating the project without the Pool Data Connections option checked. The contents of this unit, however, are very simple, and even contain IFDEFs to support CLX (for cross-platform projects, last supported by Delphi 7 and Kylix 3).

```
unit DataModuleUnit;
interface
uses
  {$IFDEF Linux}QForms, {$ELSE}Forms, {$ENDIF}
  SysUtils, Classes;

type
  TDataModule1 = class(TDataModule)
  private
  public
  end;

implementation

{$R *.dfm}

end.
```

I guess it doesn't hurt to have the Linux IFDEF in there, although IntraWeb itself no longer supports Kylix (so it has no additional benefit either).

In fact, this DatamoduleUnit is no different from one of the data modules that we could share from our existing Win32 projects (see the "Sharing VCL data modules with VCL for the Web" topic we just covered). So feel free to remove the DatamoduleUnit from your project, and just replace it with a real data module unit.

ServerController

Where the DataModuleUnit was just an example unit with an empty data module, the actual Pool Data Connection functionality is contained within the Server Controller unit. The new implementation is as follows:

```
unit ServerController;
interface
uses
  SysUtils, Classes, IWServerControllerBase, IWBaseForm, HTTPApp,
  // For OnNewSession Event
  UserSessionUnit, IWApplication, IWAppForm, DataModuleUnit, IWDataModulePool;

type
  TIWServerController = class(TIWServerControllerBase)
  Pool: TIWDataModulePool;
  procedure IWServerControllerBaseNewSession(ASession: TIWApplication;
    var VMainForm: TIWBaseForm);
  procedure IWServerControllerBaseCreate(Sender: TObject);

  procedure PoolCreateDataModule(var ADataModule: TDataModule);
  procedure PoolFreeDataModule(var ADataModule: TDataModule);
  private

  public
  end;
```

```
function UserSession: TIWUserSession;
function IWServerController: TIWServerController;
function LockDataModule: TDataModule1;
procedure UnlockDataModule(ADataModule: TDataModule1);

implementation

{$R *.dfm}

uses
  IWInit, IWGlobal;

function UserSession: TIWUserSession;
begin
  Result := TIWUserSession(WebApplication.Data);
end;

function IWServerController: TIWServerController;
begin
  Result := TIWServerController(GServerController);
end;

procedure TIWServerController.IWServerControllerBaseNewSession(
  ASession: TIWApplication; var VMainForm: TIWBaseForm);
begin
  ASession.Data := TIWUserSession.Create(nil);
end;

procedure TIWServerController.IWServerControllerBaseCreate(Sender: TObject);
begin
  Pool.Active := True;
end;

procedure TIWServerController.PoolCreateDataModule(var ADataModule: TDataModule);
begin
  ADataModule := TDataModule1.Create(nil);
end;

procedure TIWServerController.PoolFreeDataModule(var ADataModule: TDataModule);
begin
  FreeAndNil(ADataModule);
end;

function LockDataModule: TDataModule1;
begin
  Result := TDataModule1(TIWServerController(GServerController).Pool.Lock);
end;

procedure UnlockDataModule(ADataModule: TDataModule1);
var
  LTemp: TDataModule;
begin
  LTemp := ADataModule;
  TIWServerController(GServerController).Pool.Unlock(LTemp);
end;

initialization
  TIWServerController.SetServerControllerClass;
end.
```

Compared to the TIWServerController without Pool Data Connection functionality, we have a field Pool of type TIWDataModulePool and three new methods: IWServerControllerBaseCreate (so set the Pool.Active property to True) and the PoolCreateDataModule and PoolFreeDataModule methods to create and free an instance of a pooled data module.

There are two internal new functions that perform the work: LockDataModule and UnlockDataModule. These are used to ensure that only one user (session) can use a data module at the same time in the corresponding thread for that request.

The Pool component can be seen if you look at the ServerController at design-time. It has a property called PoolCount – set to 20 by default – and an Active property, set to False by default. The Active property is set to True in the IWServerControllerBaseCreate method, as mentioned before.

Using Data Pooling

In order to use the data pooling, we must call the LockDataModule function to get our hands on an instance of a pooled data module, and lock it for our use. To release it to the pool again, we need to call the UnlockDataModule, passing the data module as argument. Note that calling the UnlockDataModule must not be done too soon, otherwise rendering of your data will fail.

The best way is to call the LockDataModule function in the OnCreate event of your IntraWeb form, and the UnlockDataModule function in the OnAfterRender event of the IntraWeb form – to ensure that everything is already rendered, so it's time to release the data module to the pool again.

In Delphi code, this pattern looks as follows (with "dm" being a data module field of the IntraWeb form):

```

type
  TIWForm1 = class(TIWAppForm)
    procedure IWAppFormCreate(Sender: TObject);
    procedure IWAppFormAfterRender(Sender: TObject);
  public
    dm: TDataModule1;
  end;

```

The implementation of the OnCreate and OnAfterRender is as follows:

```

procedure TIWForm5.IWAppFormCreate(Sender: TObject);
begin
  dm := LockDataModule;
  // use dm, assign data sources
end;

procedure TIWForm5.IWAppFormAfterRender(Sender: TObject);
begin
  // un-use dm, set datasources to nil again
  UnlockDataModule(dm);
end;

```

In the OnCreate, you can start to use the data module, using the data sets or data sources on it to assign to the visual IntraWeb data-aware controls for example. And in the OnAfterRender, you can un-assign these properties (set the datasources to nil again for example) before you release the data module to the pool again.

In order for this code to compile, you must add the DatamoduleUnit as well as the ServerController unit to the uses clause of the IntraWeb form, in order to find both the LockDataModule and UnlockDataModule functions (in the ServerController unit) and the TDataModule1 type (in the DatamoduleUnit).

Custom Data Pooling

Now, let's take this example one step further and prepare the data pooling for re-used existing VCL data modules. In that case, we need to remove the `DatamoduleUnit` and replace the `TDataModule1` type with our own custom data module (say of type `TMyComplexDataModule`).

The `ServerController` needs an adjustment of the uses clause (replace `DatamoduleUnit` with the unit name of the custom data module), as well as a replacement of the `TDataModule1` type with the `TMyComplexDataModule` type. This affects the `LockDataModule`, `UnlockDataModule` as well as the `PoolCreateDataModule` and `PoolFreeDataModule` methods of the `ServerController`.

Apart from that, we also need to make the same changes to the IntraWeb form that uses the new data module. But after that's done, we can now not only share but also pool our own custom VCL data modules in VCL for the Web applications (while the same data modules are still used as normal data modules in the VCL GUI applications).

Summary

In this section I've explained and demonstrated how to use IntraWeb in Application Mode. I've covered Application Forms, the Server Controller properties and events, the `IWApplication` properties and methods, `IWAppForms`, how to navigate and pass information around, how to do session management using the `UserSession`, how to use data modules (and re-use existing data modules in your User Session), and also the IntraWeb controls from the IW Standard, IW Standard 3.2, IW Data Controls and IW Data 3.2, plus the use of Layout Managers (mainly for the HTML 3.2 pages).

I've also described how the data connection pooling works, to explain how you can add this to your existing Server Controllers (if you didn't add pooling from the start), and how to extend it by pooling custom data modules.

The next section will cover more advanced topics such as the IntraWeb Client Side support, the combination of IntraWeb with ActiveForm, AJAX support in IntraWeb applications, and finally IntraWeb Custom Components.

4. IntraWeb and AJAX

IntraWeb version 9 introduced full AJAX support to IntraWeb developers. A large number of visible IW controls now have OnAsync events that fire asynchronous and can be used to update certain parts of your page without the entire page being refreshed. Note however, that the session counter is still increased! So now more than ever, the use of the Back button is prohibited!

AJAX = Asynchronous

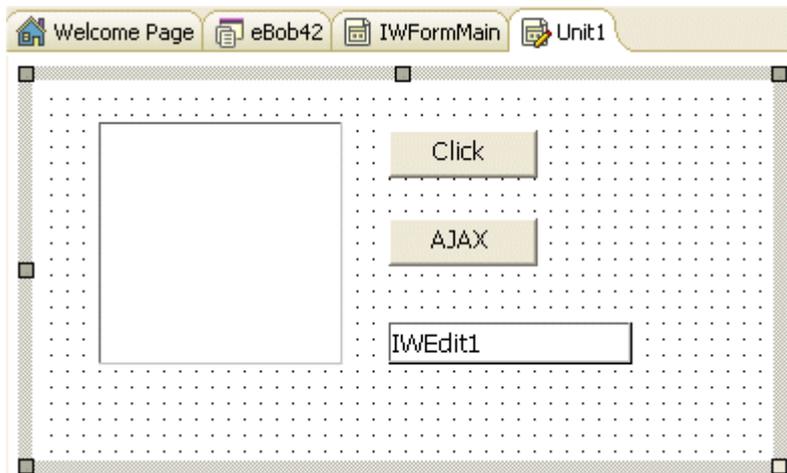
AJAX is not really a new invention, but stands for Asynchronous Javascripting And XML. Fortunately for IntraWeb developers, you really do not have to know a lot about either JavaScript or XML in order to use AJAX. What it comes down to is that an AJAX event is happening asynchronously, without the need to send a complete request or the need to refresh the entire page. This means only a partial request will be made, and a part of the page will be refreshed, resulting in a more responsive and generally more pleasant user experience (note that this works better than the previous partial update feature of IntraWeb, which is no longer covered in this manual).

IntraWeb supports AJAX events in a number of IntraWeb components through the OnAsync events and – if present – the SubmitOnAscynEvent property (by default set to True).

OnAsync

The power of AJAX is brought to us through the ease of the OnAsync event handler. For example, the TIWButton component has the OnAsyncClick and OnAsyncDoubleClick event handlers that we can implement using normal Delphi code.

As a simple example, place a TIWListbox, TIWEdit and two TIWButtons on an IntraWeb Application Form.

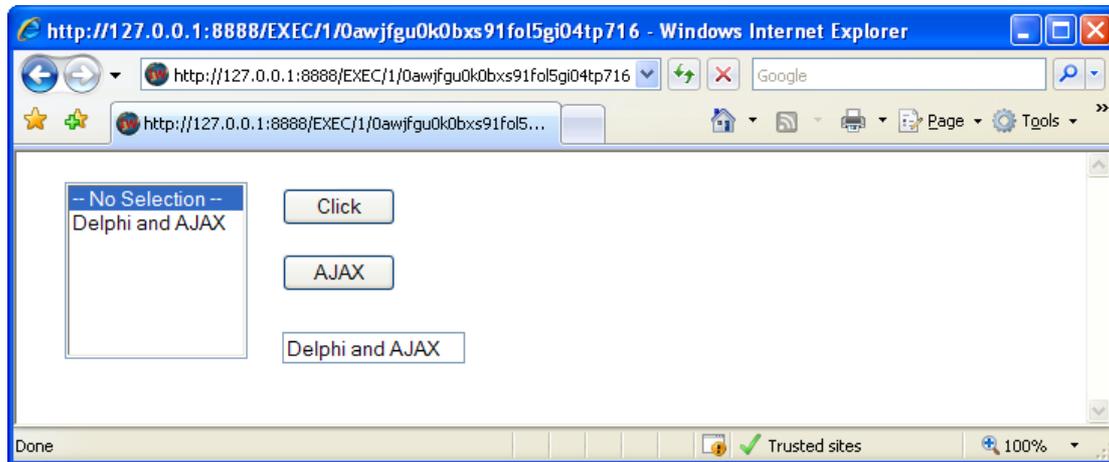


One button has a caption of Click, and implements the normal OnClick event, while the other is given a Caption of AJAX, and implements the OnAsyncClick event:

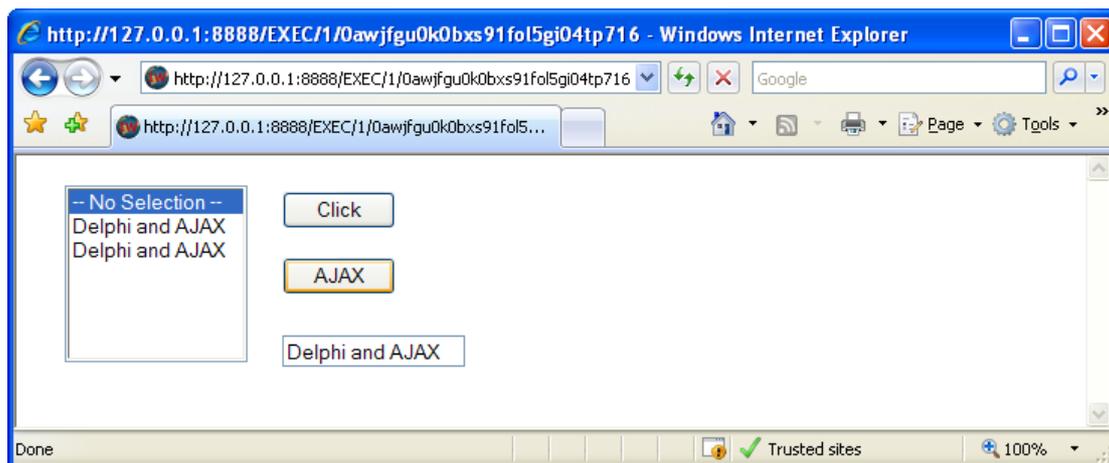
```
procedure TIWForm1.IWButton1Click(Sender: TObject);
begin
  IWListbox1.Items.Add(IWEdit1.Text)
end;

procedure TIWForm1.IWButton2AsyncClick(Sender: TObject; EventParams: TStringList);
begin
  IWListbox1.Items.Add(IWEdit1.Text)
end;
```

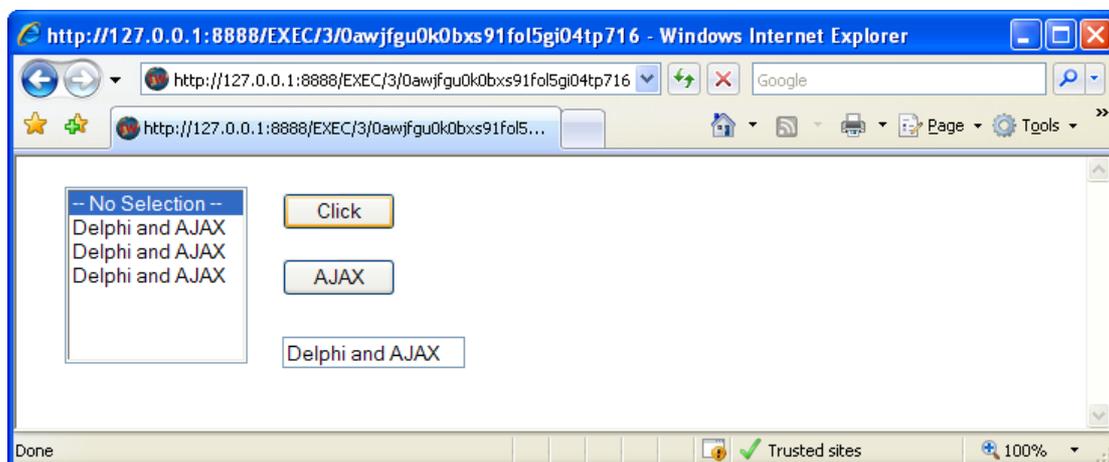
If we now enter "Delphi and AJAX" and click on the Click button, the listbox will contain the string "Delphi and AJAX" and the session counter will have the value "1" (right after the EXEC part of the URL):



If we click on the AJAX button, then the contents of the IWEdit will again be added to the IWListbox, but there will be no full request to the server. As a result, the session counter in the browser will stick at 1, but at the server it's already at 2.



We can prove this, by clicking on the Click button again, which will send a full request to the server, and add the text to the listbox once again, but also returns with the session counter of 3 (and not 2, since it already was 2 after the AJAX event).



OnAsync Events

Not all components from the IW Standard category support OnAsync events.

The TIWButton control supports 4 AJAX events: OnAsyncClick, OnAsyncDoubleClick, OnAsyncEnter and OnAsyncExit.

The TIWCheckBox and TIWRadioButton controls support 12 AJAX events: OnAsyncChange, OnAsyncClick, OnAsyncEnter, OnAsyncExit, OnAsyncKeyDown, OnAsyncKeyPress, OnAsyncKeyUp, OnAsyncMouseDown, OnAsyncMouseMove, OnAsyncMouseOut, OnAsyncMouseOver, and OnAsyncMouseUp.

TIWComboBox, TIWListBox and TIWMemo support 14 AJAX events: OnAsyncChange, OnAsyncClick, OnAsyncDoubleClick, OnAsyncEnter, OnAsyncExit, OnAsyncKeyDown, OnAsyncKeyPress, OnAsyncKeyUp, OnAsyncMouseDown, OnAsyncMouseMove, OnAsyncMouseOut, OnAsyncMouseOver, OnAsyncMouseUp and OnAsyncSelect.

The TIWEdit and TIWTimeEdit controls support 13 AJAX events: OnAsyncChange, OnAsyncClick, OnAsyncDoubleClick, OnAsyncEnter, OnAsyncExit, OnAsyncKeyDown, OnAsyncKeyPress, OnAsyncKeyUp, OnAsyncMouseDown, OnAsyncMouseMove, OnAsyncMouseOut, OnAsyncMouseOver, and OnAsyncMouseUp.

TIWImage, TIWImageFile and TIWImageButton support 6 AJAX events: OnAsyncClick, OnAsyncMouseDown, OnAsyncMouseMove, OnAsyncMouseOut, OnAsyncMouseOver, and OnAsyncMouseUp.

The TIWLabel control supports 6 AJAX events: OnAsyncClick, OnAsyncMouseDown, OnAsyncMouseMove, OnAsyncMouseOut, OnAsyncMouseOver, and OnAsyncMouseUp.

TIWLink supports only one AJAX event: OnAsyncClick.

The TIWTimer control supports only one AJAX event: OnAsyncTimer.

The TIWTabControl supports only one AJAX event: OnAsyncChange.

While the list of OnAsync events may be useful, a more useful overview is the list of the different OnAsync events as well as the components (from the IW Standard category) that support them.

AJAX Events	IW Standard Components
OnAsyncChange	TIWCheckBox / TIWRadioButton TIWComboBox / TIWListBox / TIWMemo TIWEdit / TIWTimeEdit TIWTabControl
OnAsyncClick	TIWButton TIWCheckBox / TIWRadioButton TIWComboBox / TIWListBox / TIWMemo TIWEdit / TIWTimeEdit TIWImage / TIWImageFile / TIWImageButton TIWLabel TIWLink
OnAsyncDoubleClick	TIWButton TIWComboBox / TIWListBox / TIWMemo TIWEdit / TIWTimeEdit
OnAsyncEnter	TIWButton TIWCheckBox / TIWRadioButton TIWComboBox / TIWListBox / TIWMemo TIWEdit / TIWTimeEdit
OnAsyncExit	TIWButton TIWCheckBox / TIWRadioButton TIWComboBox / TIWListBox / TIWMemo TIWEdit / TIWTimeEdit
OnAsyncSelect	TIWComboBox / TIWListBox / TIWMemo
OnAsyncTimer	TIWTimer

AJAX Key Events	IW Standard Components
OnAsyncKeyDown	TIWCheckBox / TIWRadioButton TIWComboBox / TIWListBox / TIWMemo TIWEdit / TIWTimeEdit
OnAsyncKeyUp	TIWCheckBox / TIWRadioButton TIWComboBox / TIWListBox / TIWMemo TIWEdit / TIWTimeEdit
OnAsyncKeyPress	TIWCheckBox / TIWRadioButton TIWComboBox / TIWListBox / TIWMemo TIWEdit / TIWTimeEdit
AJAX Mouse Events	IW Standard Components
OnAsyncMouseDown	TIWCheckBox / TIWRadioButton TIWComboBox / TIWListBox / TIWMemo TIWEdit / TIWTimeEdit TIWImage / TIWImageFile / TIWImageButton TIWLabel
OnAsyncMouseUp	TIWCheckBox / TIWRadioButton TIWComboBox / TIWListBox / TIWMemo TIWEdit / TIWTimeEdit TIWImage / TIWImageFile / TIWImageButton TIWLabel
OnAsyncMouseMove	TIWCheckBox / TIWRadioButton TIWComboBox / TIWListBox / TIWMemo TIWEdit / TIWTimeEdit TIWImage / TIWImageFile / TIWImageButton TIWLabel
OnAsyncMouseOver	TIWCheckBox / TIWRadioButton TIWComboBox / TIWListBox / TIWMemo TIWEdit / TIWTimeEdit TIWImage / TIWImageFile / TIWImageButton TIWLabel
OnAsyncMouseOut	TIWCheckBox / TIWRadioButton TIWComboBox / TIWListBox / TIWMemo TIWEdit / TIWTimeEdit TIWImage / TIWImageFile / TIWImageButton TIWLabel

For the components in the IW Data Controls category a similar distribution of OnAsync events exists. Note that the TIWDBNavigator and TIWDBGrid do not support any OnAsync events, however.

The components in the IW Standard 3.2 or IW Data 3.2 also do not support OnAsync events (due to the lack of the XMLHttpRequest support inside the HTML 3.2 browser).

EventParams

All OnAsync events have the following signature:

```
procedure TIWForm.IWControl1AsyncXXX(Sender: TObject;
    EventParams: TStringList);
begin

    end;
```

This means that all information will be passed to the event handler inside the EventParams argument. The TStringList contains name=value pairs, including in all cases the callback (the name of the event called) and often also a x and y coordinate, a which value (the key or mouse button pressed) and optionally modifiers.

The following table shows the different name=value pairs that are passed inside the EventParams for the different OnAsync events. Note that this list may change in different versions of IntraWeb (and please let me know if I've missed one or more).

AJAX Event	EventParams fields
OnAsyncChange	IWCONTROLNAME= callback=
OnAsyncClick	callback= x= y= which= modifiers=
OnAsyncDoubleClick	callback= x= y= which= modifiers=
OnAsyncEnter	IWCONTROLNAME= callback=
OnAsyncExit	IWCONTROLNAME= callback=
OnAsyncSelect	IWCONTROLNAME= Callback=
OnAsyncTimer	callback= AjaxRequestUniqueId=

When I write IWCONTROLNAME= this means that the complete text value of the control, or the selected index is returned as value for the name of the control (like IWEDIT1=Hello, world or IWLSTBOX1=2). This value is only present if the SubmitOnAscynEvent property it set to True.

AJAX Key Events	EventParams fields
OnAsyncKeyDown	IWCONTROLNAME= callback= x= y= which= modifiers=
OnAsyncKeyUp	IWCONTROLNAME= callback= x= y= which= modifiers=
OnAsyncKeyPress	IWCONTROLNAME= callback= x= y= which= modifiers=

The x and y contain the coordinates on screen, and the *which* key contains the ASCII value of the key that was pressed.

The value of modifiers can be CTRL_MASK, ALT_MASK, or SHIFT_MASK (the commas for CTRL and ALT are included!). Pressing the right Alt key (with the "Alt Gr" caption) has the effect that ALT_MASK,CTRL_MASK, is given, which may not be entirely correct.

AJAX Mouse Events	EventParams fields
OnAsyncMouseDown	callback= x= y= which= modifiers=
OnAsyncMouseUp	callback= x= y= which= modifiers=
OnAsyncMouseMove	callback= x= y=
OnAsyncMouseOver	callback= x= y=
OnAsyncMouseOut	callback= x= y=

The values of x and y are the screen coordinates where the mouse was present when the OnAsync event was triggered. The modifiers value can be ALT_MASK, CTRL_MASK, or SHIFT_MASK again. The ALT_MASK and CTRL_MASK include a comma, the SHIFT_MASK does not.

Working with EventParams

In order to retrieve the contents of the EventParams, I'm using the following code in my IntraWeb forms. First, I've defined a number of constants, so I don't have to retype the callback, x, y, which and modifiers strings (and make accidental typing mistakes):

```
const
  callback = 'callback';
  x = 'x';
  y = 'y';
  which = 'which';
  modifiers = 'modifiers';
```

Now, inside an OnAsync event, for example the OnAsyncClick event I can write the following code to get the X and Y coordinates:

```
procedure TIWForm2.IWEdit2AsyncKeyPress(Sender: TObject;
  EventParams: TStringList);
begin
  IWMemol.Text := 'Key [' +
    Chr(StrToIntDef(EventParams.Values[which],32)) +
    ']' at (' + EventParams.Values[x] +
    ', ' + EventParams.Values[y] + ')';
end;
```

This will give you a simple example how to get your hands on the fields of the EventParams stringlist.

Note that you cannot combine the OnAsyncXXX with the normal OnXXX events: the OnAsyncXXX will "hide" the normal events (like the OnClick which will no longer fire if you've implemented an OnAsyncClick event handler).

OnAsync and Visible

There are a few special circumstances you have to be aware of when using the OnAsync events of the IntraWeb controls. One of the special cases is the fact that invisible controls that are not rendered, cannot be made visible in an OnAsync event.

So, if you have a TIWRegion for example, with the Visible property set to False, and you want to make it visible as the result of an OnAsyncClick event, the following will not work:

```
procedure TIWForm2.IWButton3AsyncClick(Sender: TObject;  
    EventParams: TStringList);  
begin  
    IWRegion1.Visible := True;  
end;
```

The problem is that the TIWRegion was already invisible, and by default not rendered. You will need a real OnClick event to render the control.

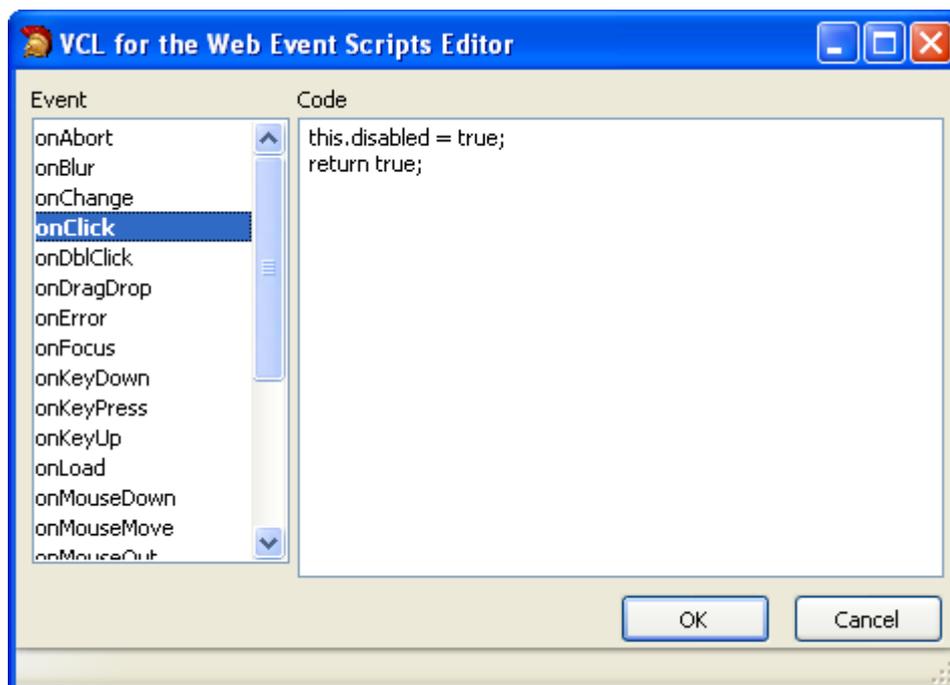
Once a control is rendered, it can be made invisible and visible again (that's no problem), but the problem is that you cannot show something which isn't there in the first place.

The really simple solution to this problem is to set the RenderInvisibleControls property of the IntraWeb Form or the Region (container) to true. This will ensure that the control on the form or region is rendered (even when invisible), so the OnAsync event can make it visible again without having to refresh the entire page.

OnAsync and Disable

If you have a TIWButton which retrieves some data from a database and updates a number of IntraWeb controls asynchronously inside the OnAsyncClick event, then you may want to disable the TIWButton or at least prevent it from sending another asynchronous OnAsyncClick event to the server.

This can be done, but requires two steps: first you need to disable the TIWButton right after the click, which can be done using JavaScript in the onClick event using the ScriptEvents:



This will disable the button as soon as you click on it.

The obvious second step involves enabling the button again. The problem is that the IntraWeb application at the server side still believes that the button is enabled (since it was a local JavaScript event that disabled the button), so just setting the Enabled property of the button to True will not have the desired effect.

In order to “trick” IntraWeb, we have to explicitly set the enabled property to False (which has no effect) and then back to True again, so the button will be enabled again.

In order to simulate a big task, let's perform a Sleep(4242) and simulate the situation in the following OnAsyncClick event:

```
procedure TIWForm2.IWButton6AsyncClick(Sender: TObject;
  EventParams: TStringList);
begin
  try
    // do some work...
    Sleep(4242);
  finally
    (Sender as TIWButton).Enabled := False;
    (Sender as TIWButton).Enabled := True;
  end;
end;
```

Summary

In this section I've discussed the way IntraWeb implements and supports AJAX asynchronous event handlers in a very elegant and straightforward way using OnAsync events with EventParams that contain more details about the event itself.

5. IntraWeb and iPhone / iPad

IntraWeb developers are not limited to using only the IntraWeb components that are delivered with the product. There are also third-party vendors like TMS Software and Arcana (which has been turned into Open Source in 2008).

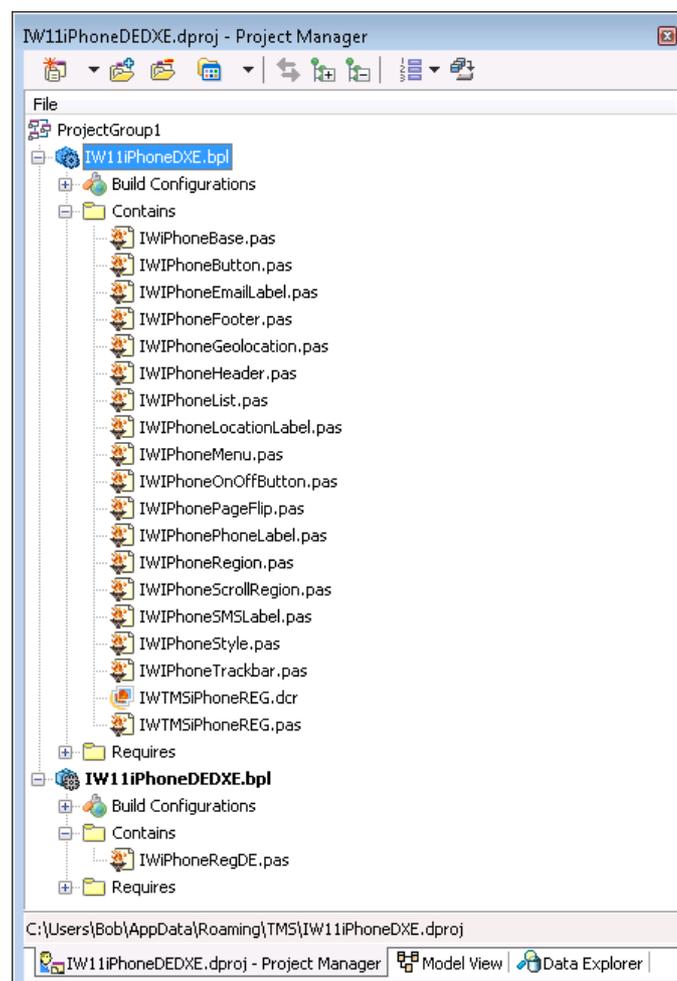
TMS Software (<http://www.tmssoftware.com>) offers a number of IntraWeb components, including a special TMW IntraWeb iPhone controls pack, which we'll cover here for iPhone and iPad web application development.

TMS IntraWeb iPhone Controls Pack

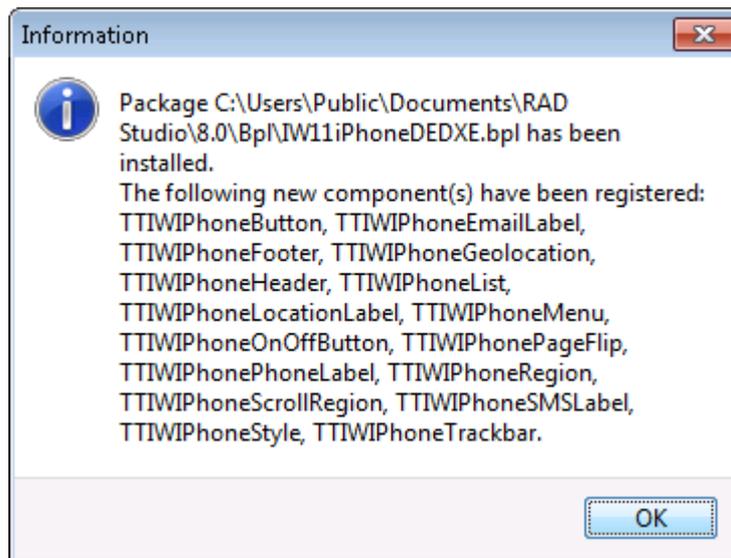
For this section, I've used version v1.5.0.0 (Jan 27, 2011) of the TMS IntraWeb iPhone Controls pack, available from <http://www.tmssoftware.com/site/tmsiiphone.asp>. These controls can work with IntraWeb 10.0.x and 11.0.x, and work with Delphi 7, 2006, 2007, 2009, 2010 and XE. Obviously, I'll use them with IntraWeb XI in Delphi XE here.

If you have purchased and downloaded the registered version of the TMS IntraWeb iPhone Controls pack, you need to unzip the iwiphonereg.zip file and place it in a directory, like C:\Users\Bob\AppData\Roaming\TMS in my case (right next to the IntraWeb XI directory in the same location).

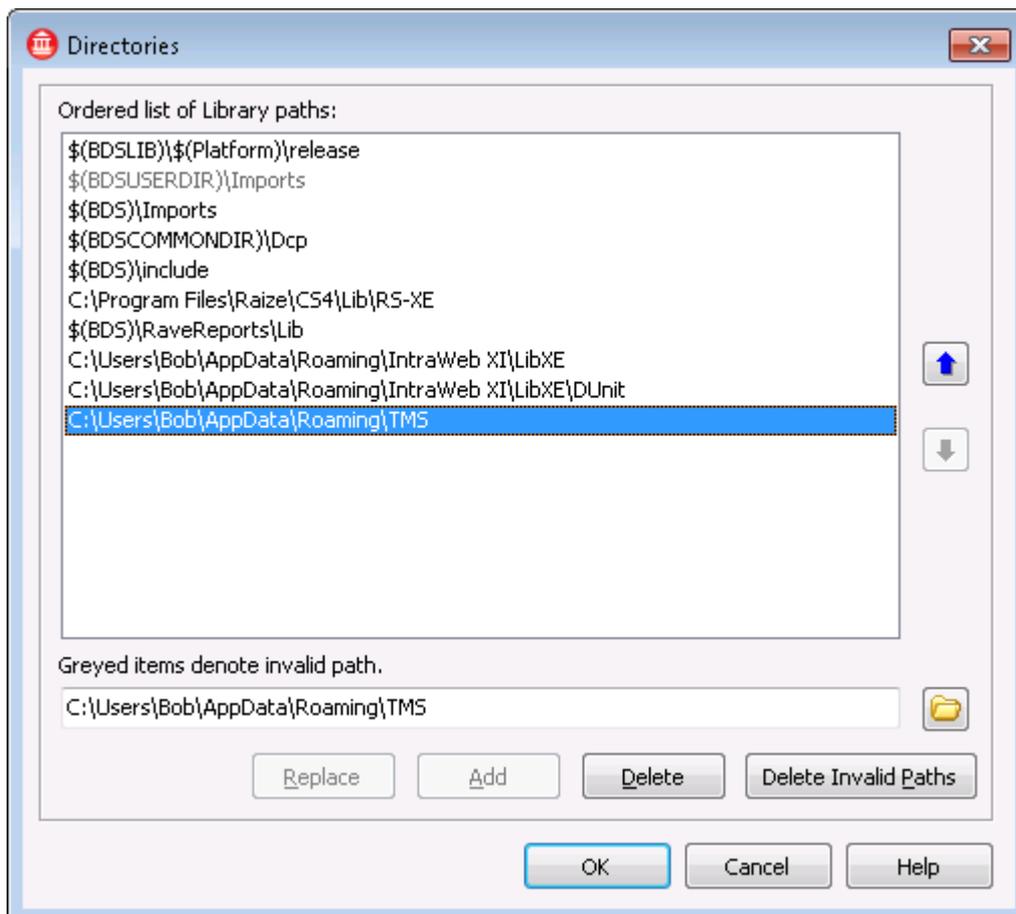
It's important to copy tmsdefs11.inc to tmsdefs.inc, otherwise you will get an error message shortly that the tmsdefs.ini file cannot be found. For Delphi XE, we then need to compile the IW11iPhoneDXE.dproj run-time package and compile and install the IW11iPhoneDEDXE.dproj design-time package.



The installation should show a list of new components:

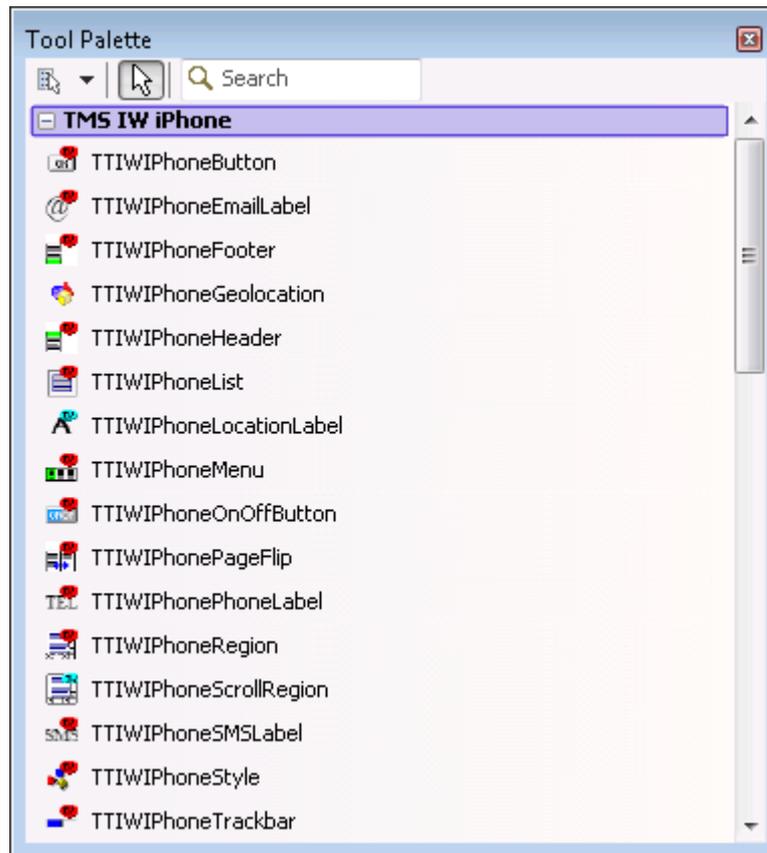


Finally, make sure to set add the directory of the TMS IntraWeb iPhone controls to your Library search path using the Tools | Options dialog of Delphi XE. In the Options dialog, select the Environment Options, Delphi Options, Library page, and then use the little button on the right of the Library Path to add the directory with the TMS source files to it, as follows:



Specifying the new search path here ensures that you do not have to update all projects.

The TMS IW iPhone components can be found in the Tool Palette of Delphi XE with the following alphabetical list of component names and icons:



All components have the TTIWiPhone prefix in their classname. Below, I will first give a short description of each of these iPhone controls, before building a first little demo application showing these iPhone controls in practice.

After the first demo, we'll see some more-or-less useful example applications for the iPhone, like the list of registered users for an event, my weblog and the game of tic-tac-toe.

TTIWiPhoneButton

The TTIWiPhoneButton is a button in the iPhone style, with optionally rounded corners, which can respond to synchronous or asynchronous click events. Apart from Round, the button can also be Square or in the shape of a Back button. The TTIWiPhoneButton can respond to the OnButtonClick but also the asynchronous OnAsyncButtonClick events.

TTIWiPhoneEmailLabel

The TTIWiPhoneEmailLabel is a special label control that will turn into a "hot" hyperlink if we assign values to the EmailAddress and optionally CCAddress, BCCAddress, Subject and Body properties. When you click on it during runtime, the standard iPhone e-mail application will be started, and all specified property values will be passed on to the e-mail application.

TTIWiPhoneFooter

The TTIWiPhoneFooter is a special footer control for the bottom of your iPhone form, which can hold a footer caption and optionally two images (left and right), for which you can write click events.

TTIWiPhoneGeolocation

The TTIWiPhoneGeolocation component can be used to determine the current location of the iPhone. It has one useful event: OnAsyncLocationRetrieved that can be used to provide feedback if the location is retrieved. Warning: if you have no or limited internet connection, then using this component will show down the rendering of the page!

TTIWiPhoneHeader

The TTIWiPhoneHeader is a special header control for the top of your iPhone form, which can contain a caption and two buttons (left and right), for which you can write click events, for either synchronously or asynchronously button clicks.

TTIWiPhoneList

The TTIWiPhoneList is a list control in the iPhone style (in standard list mode and settings mode), where each item can have an image, a value, a main caption and a subtext (notes). We can asynchronously insert or delete items from the list, and respond to click events in a synchronous and asynchronous way using OnItemClick, OnImageClick as well as OnAsyncItemClick and OnAsyncImageClick. Apart from these click events, there are also OnAsyncLoadExtraItems, OnAsyncExtraItemsLoaded, and OnRenderListItem events.

The individual items of type TiPhoneListItem have Caption, Data (TStringList), Detail, Image, Name, Notes, Section and Value properties. The events are triggered at the parent side, passing the ItemIndex of the item or image that was clicked.

The detail information for the items can be shown using the TTIWiPhonePageFlip component.

TTIWiPhoneLocationLabel

The TTIWiPhoneLocationLabel is a special label control that will turn into a “hot” hyperlink if we assign values to the predefined Location (Latitude and Longitude) and optionally Destination. When you click on it during runtime, the iPhone maps application will be started, showing your current location and optionally the destination.

TTIWiPhoneMenu

The TTIWiPhoneMenu is a special footer control that can contain a collection of menu items with images and text, plus the iPhone style status indicators. We can update the status of the menu items, and respond to a user click, both synchronous and asynchronous.

The Items property of the TTIWiPhoneMenu control holds the TiPhoneMenuItem which each contain properties for Caption, Image, IndicatorCaption, IndicatorVisible, Name, and SelectedImage.

The individual items do not have a click event of their own, but the parent TTIWiPhoneMenu has the OnItemClick or OnAsyncItemClick events, passing the ItemIndex of the button that was clicked (starting to count at 0).

TTIWiPhoneOnOffButton

The TTIWiPhoneOnOffButton is a special iPhone style button with an “On” mode and an “Off” mode, that can be displayed in three styles: normal, system and custom. We can use the asynchronous events to update the button and to respond to the click events. Using the OffCaption and OnCaption you can translate the Off and On captions on this button.

The actual state of the TTIWiPhoneOnOffButton can be retrieved or set using the ButtonState property, which can be bsOff or bsOn.

TTIWiPhonePageFlip

The TTIWiPhonePageFlip is a control that can be used to flip between TIWiPhoneRegion controls, featuring asynchronous updates. Ideal in combination with the TIWiPhoneList component to provide details for a list item.

The TTIWiPhonePageFlip has a FrontRegion and a BackRegion property. Both can be assigned to a TWIWiPhoneRegion control. Switching between the front region and the back region can be done using the SlideToFront and SlideToBack methods of the TTIWiPhonePageFlip, or by assigning arFront or arBack to the ActiveRegion property.

The transition can be controlled using AnimationSpeed and AnimationType (which can be set to atDrop, atFlip, atSlide, atSwing or atTurn).

TTIWiPhonePhoneLabel

This is a special label control that will turn into a “hot” hyperlink if we assign a value to the TelephoneNumber property. When you click on it during runtime, the iPhone will make a call to the specified number.

TTIWiPhoneRegion

A region component that we can use to place IntraWeb and TMS iPhone controls, which automatically shows itself at the correct iPhone, iPod and iPad screen sizes.

The TTIWiPhoneRegion has properties for AutoClientAlignAtRuntime, ClipRegion, Device (which can be set to iPhone or iPad), DeviceOrientation (oVertical or oHorizontal, although the iPhone and iPad will automatically cause the orientation to switch),

TTIWiPhoneScrollRegion

This is a special TIWiPhoneRegion component that offers iPhone style scrolling and scroll indicator, and also keeps the Header and Footer or Menu fixed, so only the content of the Region itself will scroll.

Note that if you want to display a scrollable list of items, you should consider using a TTIWiPhoneList on a TTIWiPhoneRegion instead of using a TTIWiPhoneScrollRegion.

TTIWiPhoneSMSLabel

This is a special label control that will turn into a “hot” hyperlink if we assign a value to the SMSNumber property. When you click on it during runtime, the iPhone will start an SMS message (initially empty) to the specified number.

TTIWiPhoneStyle

The TWITiPhoneStyle component is responsible for the style of the iPhone/iPad application, and has a number of useful properties that we should set.

The BGColor and BGColorTo properties, default both set to \$00D4CCC5 can be used to set the gradient background color of the page.

The StartupImage can contain the URL for a splash screen that will be used on the iPhone or iPad when the application starts. Even more useful is the IconImage property which can be set to the URL of the icon that’s used if we place the IntraWeb application on the home screen (desktop) of the iPhone. Note that for best results you should use PNG files, and specify a URL that can be found by the client (i.e. do not use localhost). The images will be resolved when the application is started and cached (so if you change the remote image, it will take another startup for the splash screen to be changed for example).

TTIWiPhoneTrackbar

The TIWiPhoneTrackbas is an iPhone style horizontal trackbar, like a slider control, which can be updated asynchronously, and can produce both synchronous OnEndDrag and asynchronous events OnAsyncStartDrag, OnAsyncDrag and OnAsyncEndDrag.

TMS iPhone Controls Demo

In order to show the TMS IntraWeb iPhone Controls in action, we should create a new VCL for the Web (IntraWeb) application, and place a `TIWiPhoneStyle` control on the form to ensure that the correct iPhone/iPad styles are used. You can change the `BGColor` (top) and `BGColorTo` (bottom) colours from the default `$00D4CCC5` to something else like `clSkyBlue` for the `BGColorTo`.

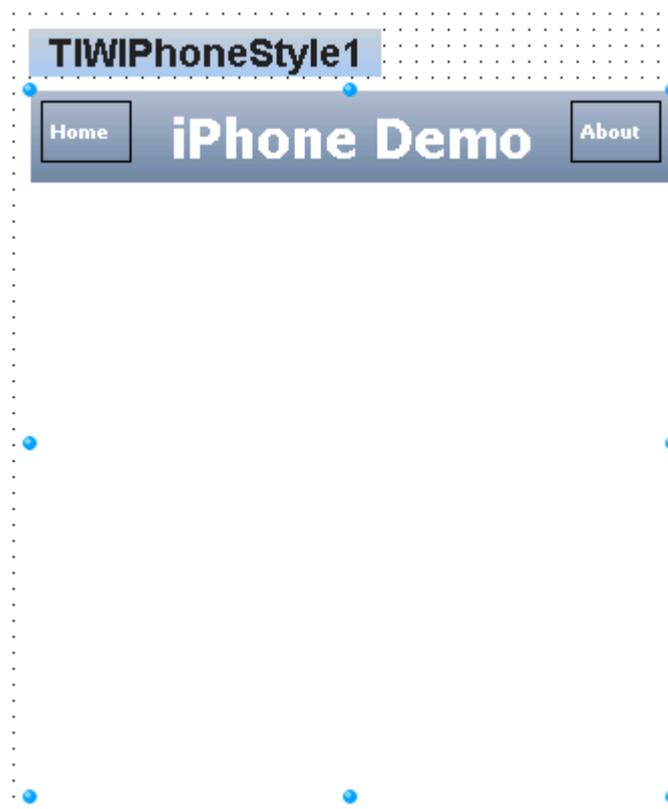


You should also set the `IconImage` and `StartupImage` in case you want other people to use your application and create a shortcut on their iPhone desktop.

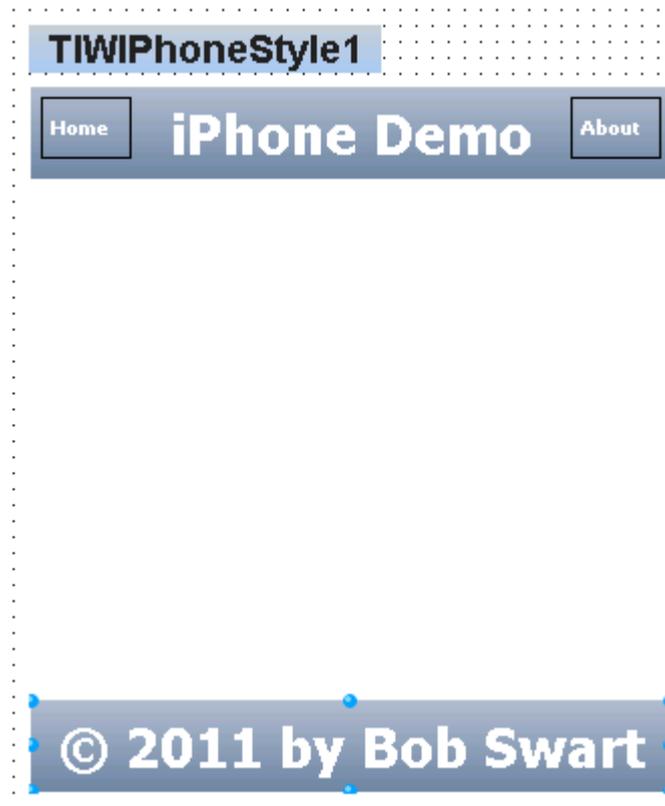
Next, you should place a `TIWiPhoneRegion` on the form, which will show us the correct height (355) and width (320) of the browser page in the iPhone or iPod.

Note that there is currently a repaint bug in Delphi XE with IntraWeb XI and the `TIWiPhoneRegion` component: when you resize the form, the `TIWiPhoneRegion` will become invisible (normally, it's white, but when you resize the form, the form designer grid will be shown in the location of the `TIWiPhoneRegion` as well). Just click on where you think the `TIWiPhoneRegion` is to select it and show up as a white region again.

With the `TIWiPhoneRegion` component selected, place a `TIWiPhoneHeader` component on the `TIWiPhoneRegion`. It will automatically be positioned at the top of the `TIWiPhoneRegion`. You can configure the `TIWiPhoneHeader`, for example by changing the `Caption` to "iPhone Demo" (don't make the header caption too long, or it will mess up the "Home" and "About" buttons that are also placed on the `TIWiPhoneHeader`).



With the TIWiPhoneRegion component selected, also place a TIWiPhoneFooter component on the TIWiPhoneRegion. It will automatically be positioned at the bottom of the TIWiPhoneRegion. You can configure the TIWiPhoneFooter by changing the Caption as well, for example to © 2011 by Bob Swart



Without placing any content in the middle of the TIWiPhoneRegion, we can already respond to two events: the left button and the right button on the header. The TIWiPhoneHeader component has no less than four events for them: both synchronous and asynchronous events. The synchronous events can be handled by implementing the OnLeftButtonClick and OnRightButtonClick, the more efficient AJAX-based asynchronous ones using OnAsyncLeftButtonClick and OnAsyncRightButtonClick.

As a first demo, let's implement the About button (right button click) to put the current date and time in the Caption of the TIWiPhoneFooter. This can be done as follows using the synchronous approach:

```
procedure TIWForm1.TIWiPhoneHeader1RightButtonClick(Sender: TObject);  
begin  
    TIWiPhoneFooter1.Caption := DateTimeToStr(Now)  
end;
```

Clicking on the right button with this event will cause the entire page to refresh, which is not always a good idea.

The asynchronous approach is implemented as follows (make sure to remove the implementation of the synchronous event, as you cannot have both at the same time):

```
procedure TIWForm1.TIWiPhoneHeader1AsyncRightButtonClick(Sender: TObject;  
    EventParams: TStringList);  
begin  
    TIWiPhoneFooter1.Caption := DateTimeToStr(Now)  
end;
```

Using the iPhone Simulator on the Mac, we can see the application with the Header, Footer and two buttons in the header. Clicking on the About button on the right will put the current date and time in the footer, just as expected:

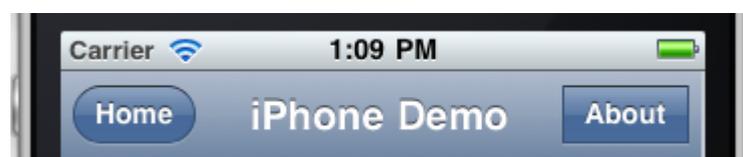


I leave it as exercise for the reader to compare the behaviour of the synchronous approach (with a full screen refresh) to the asynchronous version.

Note that the TIWiPhoneFoot can also have two buttons: one on the left and one on the right. The TIWiPhoneHeader has LeftButton and RightButton properties, consisting of Appearance, ButtonType, Caption, Font, ImageURL (so you can place small images on the buttons as well) and the Visible property to control if they are shown or not.

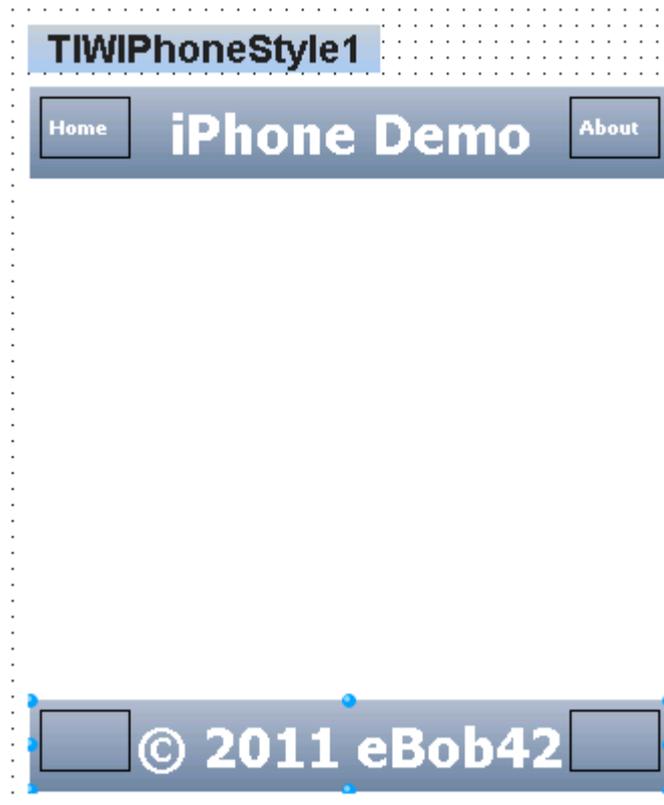
The Appearance subproperty of the LeftButton and RightButton can control the BGColor and BGColorTo, the disabled colour, the "hot" colour, mirror colour, and a different colour for each of the borders (top, bottom, leftright, and disabled).

The ButtonType can be set to btBack (the left-pointing shape you see as the "Home" button in the above screenshots), btDefault (used for the About button on the right), btRound, or btSquare. Changing the ButtonType will have no effect at design time, but below you can see the btRound (at the left side) and btSquare (at the right side) on the Header of the page.



The TIWiPhoneFooter does not have a LeftButton and RightButton property, but it offers a LeftImageURL and a RightImageURL property together with a similar set of click events: OnLeftImageClick, OnRightImageClick and the asynchronous counterparts OnAsyncLeftImageClick and OnAsyncRightImageClick.

The button images will not appear at design-time, but an area will appear to indicate where they will be shown on the footer (so you may want to decrease the size of the font of the TIWiPhoneFooter or shorten the Caption).



In the OnLeftImageClick and OnRightImageClick events (not the asynchronous ones) you can also change the LeftImageURL and RightImageURL for a dynamic effect:

```

procedure TIWForm1.TIWiPhoneFooter1LeftImageClick(Sender: TObject);
begin
  if TIWiPhoneFooter1.LeftImageURL = 'http://192.168.62.42/star_s.png' then
    TIWiPhoneFooter1.LeftImageURL := 'http://192.168.62.42/star.png'
  else TIWiPhoneFooter1.LeftImageURL := 'http://192.168.62.42/star_s.png'
end;

```

Of course, you should also do some “real” work in these event handlers, but you get the idea.



Both the TIWiPhoneHeader and TIWiPhoneFooter have a property StatusText which will not show up at design-time, but will appear on the iPhone at runtime.

Apart from TMS iPhone controls, we can also place “regular” controls like a TIWEdit and TIWLabel for example. Place both a TIWEdit and TIWLabel on the TIWiPhoneRegion, change the Caption of the Label to “What’s your name?” and clear the Text property of the Edit. Next, place a TIWiPhoneButton component below these two, make it a bit wider, set the ButtonType property to btRound, and the Caption property to “Say Hello”. In the OnAsyncButtonClick event handler, we can use the value of the Edit to write a new caption in the Footer for example, as follows:

```
procedure TIWForm1.TIWiPhoneButton1AsyncButtonClick(Sender: TObject;  
    EventParams: TStringList);  
begin  
    TIWiPhoneFooter1.Caption := 'Hello ' + IWEdit1.Text  
end;
```

When you run this application and click on the Edit control to enter a name, the iPhone keyboard will slide-up, allowing you to use the keyboard to enter a name. If you click on the “Done” button, the keyboard will slide down again.



There are also four special label controls included in the TMS iPhone Control pack: namely TIWiPhoneEmailLabel, TIWiPhoneLocationLabel, TIWiPhonePhoneLabel and TIWiPhoneSMSLabel. These labels look like simple labels at design-time, but can perform a specific action at runtime.

The screenshot below has the four special labels placed on the TIWiPhoneRegion:



The TIWiPhoneEmailLabel component has string properties EmailAddress, CCAddress, and BCCAddress to control the people receiving the e-mail, as well as a property Subject and a TStringList property Body to contain the actual message. There are no less than 6 possible OnAsync event handlers we can use: OnAsyncClick being the most obvious, but also OnAsyncMouseDown, OnAsyncMouseMove, OnAsyncMouseOut, OnAsyncMouseOver and OnAsyncMouseUp.

Even if we do not implement any of these on Async events, if we click on the label at runtime it will automatically start the email application on the iPhone with the values we entered for the properties.

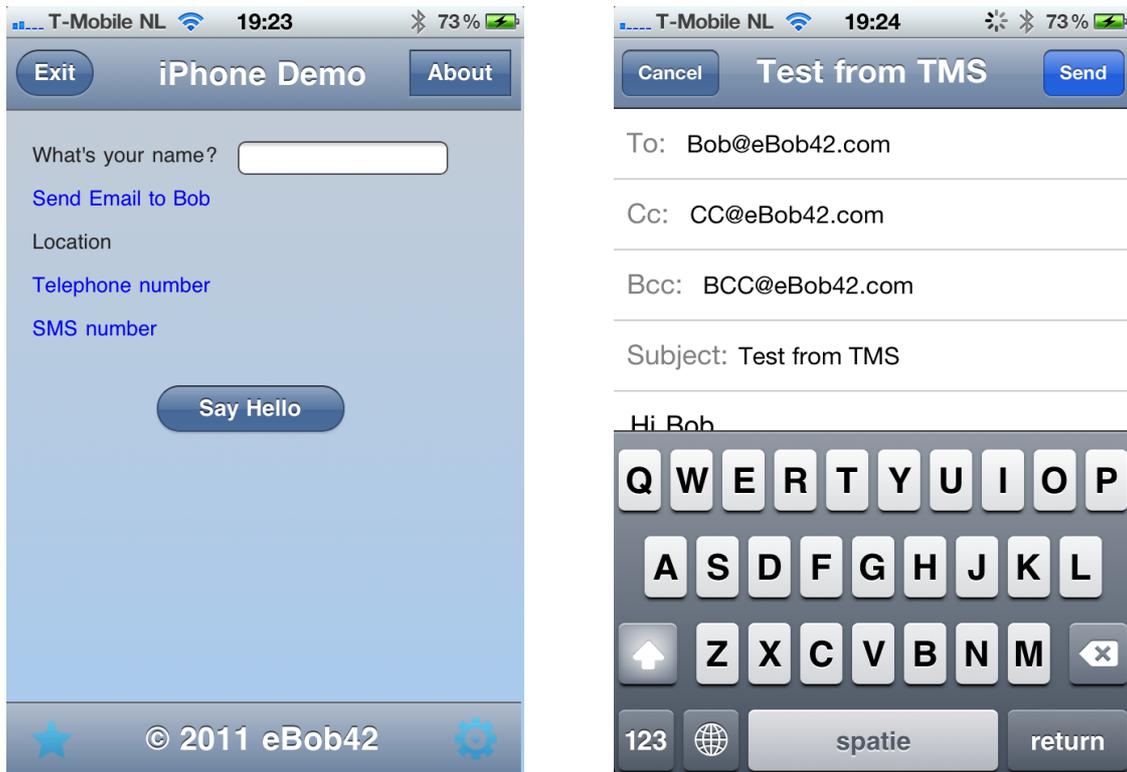
TIWiPhoneLocationLabel has Destination, DestinationLatitude, DestinationLongitude, Location, Latitude, and Longitude properties.

The TIWiPhonePhoneLabel has a TelephoneNumber property and will automatically make a phone call when you click on the label at runtime.

The TIWiPhoneSMSLabel has a SMSNumber property and will automatically start the SMS application on the iPhone to send an SMS message.

Note that neither of these special labels appear to work in the iPhone Simulator on the Mac, but they work just fine on a real iPhone. They may have limited functionality on the iPad and iPod Touch (who generally cannot phone or send SMS messages, but may be able to send e-mail messages if a wifi connection is active, and they may also be able to use the Location link).

When clicking on the Email label for example, the default email application of the iPhone will start.



Of course, we can also tilt the iPhone and allow the page to be displayed in horizontal orientation.



This will automatically happen, without any required coding from our side. Of course, it may be a good idea to make sure all controls will remain visible. In case of a normal TIWiPhoneRegion, this may not happen at all times. In that case, it may be wise to start using a TIWiPhoneScrollRegion, which is a IWiPhoneRegion control that offers iPhone style scrolling and a scroll indicator. The Header and Footer will remain fixed!

Two TMS iPhone controls that haven't been used yet are the TIWiPhoneOnOffButton and the TIWiPhoneTrackbar component. We can place them on the TIWiPhoneRegion control to view them in action. Note that the design-time look is much simpler than the runtime look. The TIWiPhoneOnOffButton component has both an OnButtonClick and an asynchronous OnAsyncButtonClick event that we can respond to. Usually, this button is used to enable or disable options in a settings page of the iPhone application. However, we can also use it to demonstrate another control that we haven't covered, yet, the TIWiPhonePageFlip component, which can be used to flip from one Region to another.

Place a TTIWiPhonePageFlip component on the form, set the align property to alClient, and then we can use the FrontRegion and BackRegion properties to assign two different TIWiPhoneRegion controls to it.

As example second region, place another TIWiPhoneRegion component on the form, and put a TIWiPhoneHeader control on it (which will automatically move to the top) followed by a TIWiPhoneMenu control.

The TIWiPhoneMenu control will be placed at the bottom of the new region, and can be used to display menu items. We can use the Items property to add a collection of TiPhoneMenuItem, each with a Caption, Image, IndicatorCaption, IndicatorVisible, and SelectedImage property. Adding three menu items, the form designer could now resemble the following layout (with the new TIWiPhoneOnOffButton, TIWiPhoneTrackbar as well as a new TIWiPhoneRegion on the right with a header and an TIWiPhoneMenu control).



In order to demonstrate the power and ease of use of the TIWiPhonePageFlip control, we can implementing the sliding between the front region and the back region. First of all, the actual sliding technique itself can be controlled by the AnomationType property, which by default is set to atSlide, but can also be atDrop, atFlip, atSwing or atTurn. The AnomationSpeed controls the speed that it takes for the animation to be shown; by default 500 ms.

In order to switch from Front to Back, we can call the TIWiPhonePageFlip1.SlideToBack method. And in a similar way, to get back to the Front, we can call the SlideToFront method.

Just for fun, we can make sure that the TIWiPhoneOnOffButton will slide to the second region when we set the button to "On", and the Back button on the Header of the second region will slide back to the first region. These two event handlers were implemented as follows:

```

procedure TIWFormBlog.TIWIPhoneHeader2AsyncLeftButtonClick(Sender: TObject;
  EventParams: TStringList);
begin
  TIWIPhonePageFlip1.SlideToFront;
end;

procedure TIWFormBlog.TIWIPhoneOnOffButton1AsyncButtonClick(Sender: TObject;
  EventParams: TStringList);
begin
  if TIWiPhoneOnOffButton1.ButtonState = bsOn then
    TIWIPhonePageFlip1.SlideToBack;
end;

```

The result can be seen below, where the left screenshot was taken just before I clicked on the "Off" button to switch it to "On" and slide the right screenshot in (which is still empty, but also shows the iPhone Menu instead of the Footer).



We can also respond to the menu item events on the second region, which is done in a single event if the menu itself, passing the ItemIndex of the button that was clicked (starting to count from 0).

```

procedure TIWFormBlog.TIWIPhoneMenuItemClick(Sender: TComponent;
  ItemIndex: Integer);
begin

end;

```

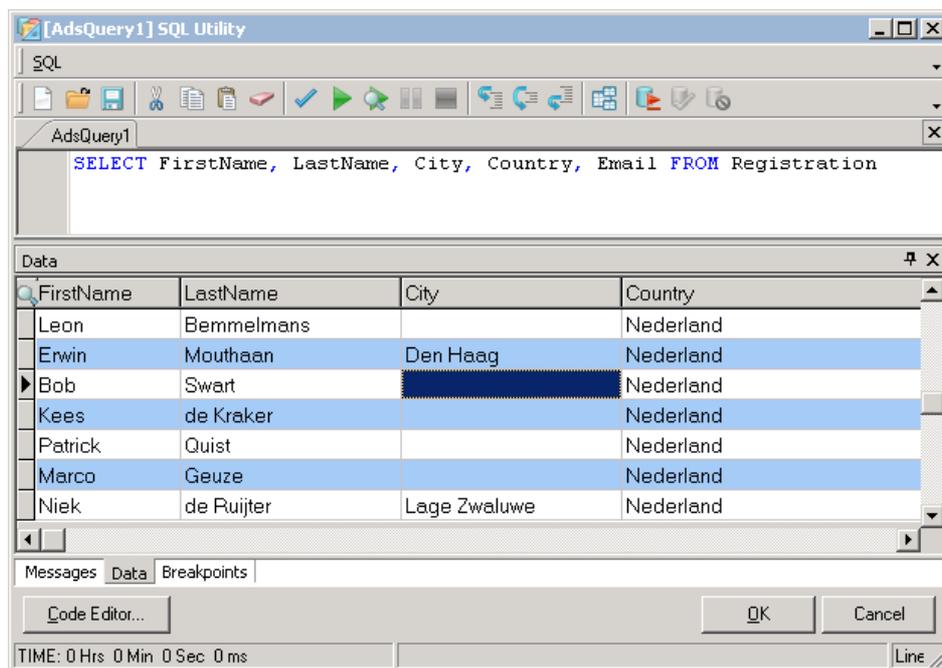
I leave it as exercise for the reader to come up with a nice implementation of the menu options here.

Registered Users

There is one last TMS iPhone control that we haven't covered, yet: the TIWiPhoneList which can show a list of items, like a listbox, but with support for not only a single line of text, but also a subtext, a number, an image and some detailed notes. Using the demo so far, we can place a TIWiPhoneList control in the second TIWiPhoneRegion and prepare to fill it with the list of registered users for an event for example.

This particular example is using the Advantage Database Server with the Registration table in the EVENT.ADD data dictionary (with alias "Events" in the ADS.INI file). So we need an TAdsConnection component to connect to the Event alias, setting AliasName to Events, and the LoginPrompt to False.

A TAdsQuery component is needed to connect its AdsConnection property, and enter the SQL command "SELECT FirstName, LastName, City, Country, Email FROM Registration", which returns the following in the SQL Editor:



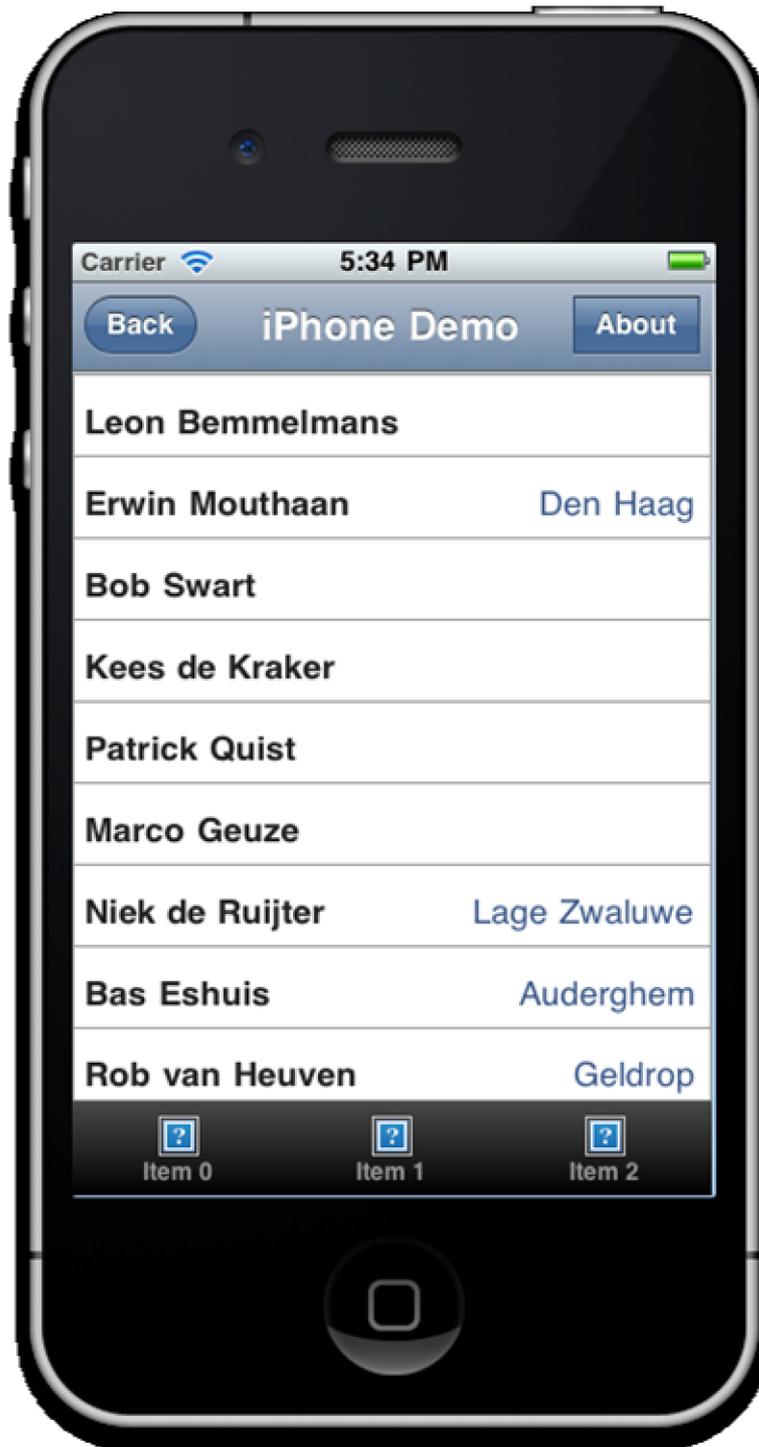
Since this list won't change while loading and presenting the form, we can build the TIWiPhoneList in the OnCreate event of the IntraWeb Form, as follows:

```

procedure TIWFormBlog.IWAppFormCreate(Sender: TObject);
begin
  ADSQuery1.Open;
  ADSQuery1.First;
  while not ADSQuery1.Eof do
  begin
    with TIWiPhoneList1.Items.Add do
    begin
      Caption := ADSQuery1.FieldName('FirstName').AsString + #32 +
        ADSQuery1.FieldName('LastName').AsString;
      Value := ADSQuery1.FieldName('City').AsString;
      // Image := DataSource1.DataSet.FieldName('Country').Value + '.png';
    end;
    ADSQuery1.Next;
  end;
end;

```

And the result, when running in the iPhone Simulator, is the list of registered users.



Another example of an IntraWeb iPhone application is the iPhone edition of my weblog which can be found at <http://www.bobswart.nl/iblog>

Summary

In this section, I've covered the TMS IntraWeb iPhone Controls pack, a third-party add-on for IntraWeb to allow us to quickly and easily create web applications in the iPhone style (but also for the iPad and iPod Touch).

The TMS IntraWeb iPhone controls pack is available as separate purchase, or as part of the full TMS IntraWeb Component Studio.

6. IntraWeb Custom Components

IntraWeb developers are not limited to using only the IntraWeb components that are delivered with the product. There are also third-party vendors like TMS and Arcana (which has been turned into Open Source in 2008).

We can also write our own IntraWeb custom components, and in this section we will examine how writing and using IntraWeb custom components can be done.

IntraWeb Controls

As an example, let's take a look at the unit IWCompLabel.pas, which contains both the TIWCustomLabel and TIWLabel components.

As you can see, the TIWCustomLabel component is derived from TIWControl, which is the parent class of all IntraWeb components. From the TIWControl, we must at least override the Create constructor as well as the RenderHTML function. The later is responsible for producing the HTML that is rendered when the component is shown inside the browser.

```
unit IWCompLabel;
interface
uses
{$IFDEF Linux}QGraphics, {$ELSE}Graphics, {$ENDIF}
{$IFDEF Linux}QControls, {$ELSE}Controls, {$ENDIF}
  Classes,
  IWControl, IWHTMLTag;

type
TIWCustomLabel = class(TIWControl)
protected
  FRawText: Boolean;
  procedure SetAutoSize(Value: Boolean); override;
public
  constructor Create(AOwner: TComponent); override;
  function RenderHTML: TIWHTMLTag; override;
  //
  property AutoSize;
  property RawText: Boolean read FRawText write FRawText default True;
published
  property Font;
end;

type
TIWLabel = class(TIWCustomLabel)
published
  property AutoSize;
  property Caption;
  property RawText;
end;

implementation
uses
  SysUtils;

function TIWCustomLabel.RenderHTML: TIWHTMLTag;
begin
  Result := TIWHTMLTag.CreateTag('SPAN');
  try
    if not RawText then
```

```

        Result.Contents.AddText(TextToHTML(Caption))
    else
        Result.Contents.AddText(Caption);
    except
        FreeAndNil(Result);
        raise
    end
end;

constructor TIWCustomLabel.Create(AOwner: TComponent);
begin
    inherited Create(AOwner);
    Height := 21;
    Width := 121;
    AutoSize := true;
    FRawText := true
end;

procedure TIWCustomLabel.SetAutoSize(Value: Boolean);
begin
    inherited SetAutoSize(Value);
    Invalidate
end;

end.

```

Note the TextToHTML method that is used in case the RawText property is set to False. Of course, the TIWLabel component also needs to be registered as component in Delphi, using the RegisterComponents function. But far more important than this last step, is the registration of the so-called paint handler for the IntraWeb control.

While the RenderHTML method is used to produce the HTML at run-time, the paint handlers give the IntraWeb components the ability to draw themselves at design-time. The unit IWDsnPaint.pas contains the base classes for the paint handlers, and the unit IWDsnPaintHandlers contains the definition for all IntraWeb paint handlers, including the one for the TIWLabel, namely TIWPaingHandlerLabel, which is derived from the base class TIWPaintHandlerDsn and defined as follows:

```

type
    TIWPaintHandlerLabel = class(TIWPaintHandlerDsn)
    public
        procedure Paint; override;
    end;

```

The implementation of the Paint method can be as follows (this is just used to illustrate how a paint method from a specific TIWPaintHandler is implemented):

```

procedure TIWPaintHandlerLabel.Paint;
var
    LWidth: Integer;
    LLabel: TIWCustomLabel;
    LLeft: Integer;
begin
    LLabel := FControl as TIWCustomLabel;
    with FControl.Canvas do
    begin
        Brush.Style := bsClear;
        if LLabel.Color <> clNone then
            Brush.Color := LLabel.Color;
        SetCanvasFont(LLabel.Font);
    end;
end;

```

```

if LLabel.Align = alNone then
begin
  if LLabel.AutoSize then
    LWidth := Trunc(TextWidth(LLabel.Caption) * 1.10)
  else
    LWidth := LLabel.Width
  end
else LWidth := LLabel.Width;
  LLeft := 0;
  if not LLabel.AutoSize then
  begin
    case LLabel.Alignment of
      taLeftJustify: LLeft := 0;
      taCenter: LLeft := (LWidth div 2) -
        (TextWidth(LLabel.Caption) div 2);
      taRightJustify: LLeft := LWidth - TextWidth(LLabel.Caption)
    end
  end;
  TextRect(Rect(LLeft,0,LWidth,TextHeight(LLabel.Caption)),
    LLeft,0,LLabel.Caption);
  if (LLabel.Align = alNone) and LLabel.AutoSize then
  begin
    LLabel.Width := LWidth;
    LLabel.Height := TextHeight(LLabel.Caption)
  end
end
end;

```

And finally, in order to "connect" the TIWPaintHandlerLabel to the TIWLabel IntraWeb component, we have to call the IWRegisterPaintHandler method in the initialization section of the unit, as follows:

```
IWRegisterPaintHandler('TIWLabel', TIWPaintHandlerLabel);
```

Custom Components

Now that we've examined the TIWLabel component, it's time to create our own custom IntraWeb components. We have basically two choices when it comes to building custom IntraWeb components: either start with an existing component, one that already has a design-time paint handler defined, or start with one of the "custom" parent classes, and make sure not to forget your own design-time paint handler.

As simple TIWCustomLabel derived component, we can do this:

```

unit IWHelloWorld;
interface
uses
  Classes, IWCompLabel;

type
  TIWHelloWorld = class(TIWCustomLabel)
  public
    constructor Create(AOwner: TComponent); override;
  end;

  procedure Register;

implementation
uses
  IWdsnPaintHandlers, IWBaseControl;

```

```

constructor TIWHelloWorld.Create(AOwner: TComponent);
begin
    inherited;
    Text := 'Hello, IntraWeb World';
end;

procedure Register;
begin
    RegisterComponents('IW Custom', [TIWHelloWorld])
end;

initialization
    IWRegisterPaintHandler('TIWHelloWorld', TIWPaintHandlerLabel);
end.

```

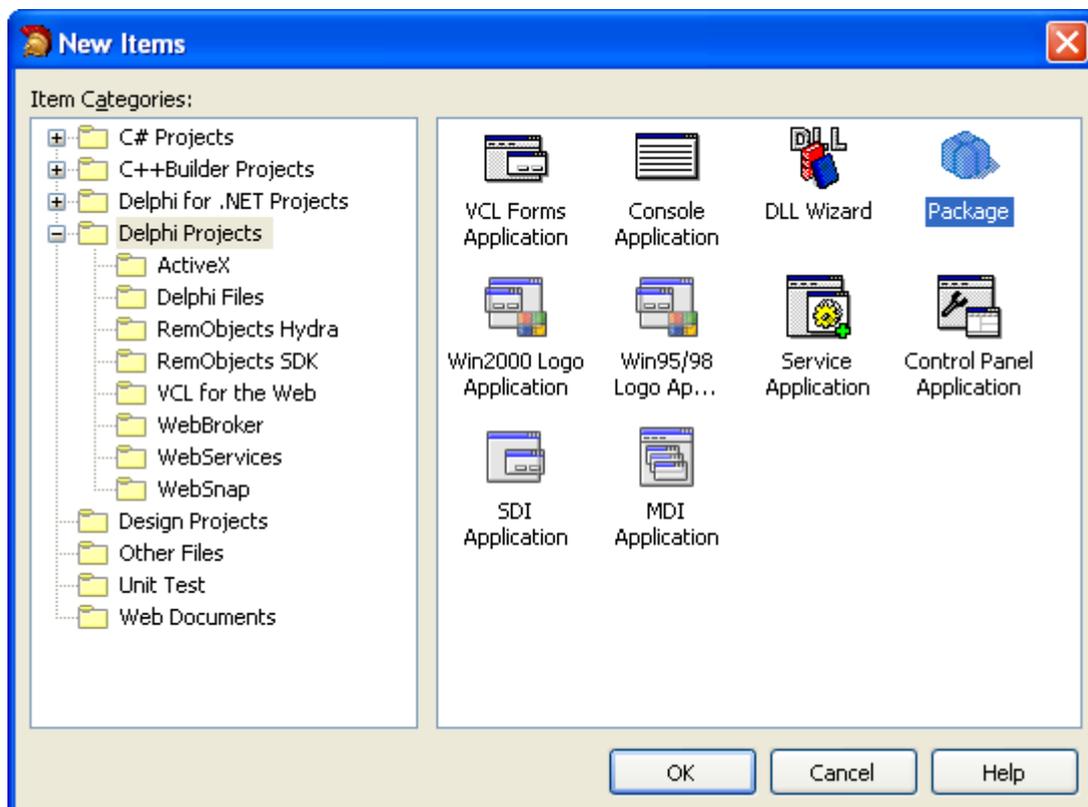
However, while this component installs fine, and can be used just fine at design-time, we'll get compiler errors (or rather linker errors) when we try to compile an application using the IWHelloWorld component.

This is caused by the fact that the design-time paint handlers should only be included at design-time, and not in the run-time code for the components. In short, our IntraWeb component source code should be split in two parts: the run-time section, and the design-time section.

Packages

In order to handle and contain the different run-time and design-time parts, we need two packages: a run-time package and an associated design-time package.

In order to start a new Delphi Package project, do *File / New – Other*, and select the Package icon from the Delphi Projects category in the Object Repository:



This will start a new package project, that we can save in eBob42IW.dproj. This package already has one item in the requires node: the rtl.dcp package. In order to derive from

existing IntraWeb components (or use existing IntraWeb types), we must also add the IntraWeb run-time package to the requires node as well. For IntraWeb XI and Delphi XE that's the IntraWeb_110_150.bpl run-time package, which can be found on your machine in the Windows\System32 directory. Note that there is also a IntraWebDB_110_150. bpl package with the database controls. Finally, there is a dclIntraWeb_110_150.bpl (in the C:\Documents and Settings\%USERNAME%\Application Data\IntraWeb XI\LibXE directory, but that's a design-time package that contains the design-time functionality.

Using the same name pattern, we can create another package – a design-time package – with the name dclBob42IW.dproj, and add the dclIntraWeb_110_150.dcp design-time package to its requires node. From now on, we can put the run-time functionality in the eBob42IW package and the design-time functionality in the dclBob42 package.

The TIWHelloWorld component can now be split into two files: TIWHelloWorld.pas for the run-time code, and TIWDsnHelloWorld.pas for the design-time support of the TIWHelloWorld component.

The unit IWHelloWorld.pas has the following contents:

```
unit IWHelloWorld;
interface
uses
  Classes, IWCompLabel;

type
  TIWHelloWorld = class(TIWCustomLabel)
  public
    constructor Create(AOwner: TComponent); override;
  end;

implementation

constructor TIWHelloWorld.Create(AOwner: TComponent);
begin
  inherited;
  Text := 'Hello, IntraWeb World';
end;

end.
```

With the design-time registration unit IWDsnHelloWorld.pas as follows:

```
unit IWDsnHelloWorld;
interface
uses
  Classes, IWHelloWorld;

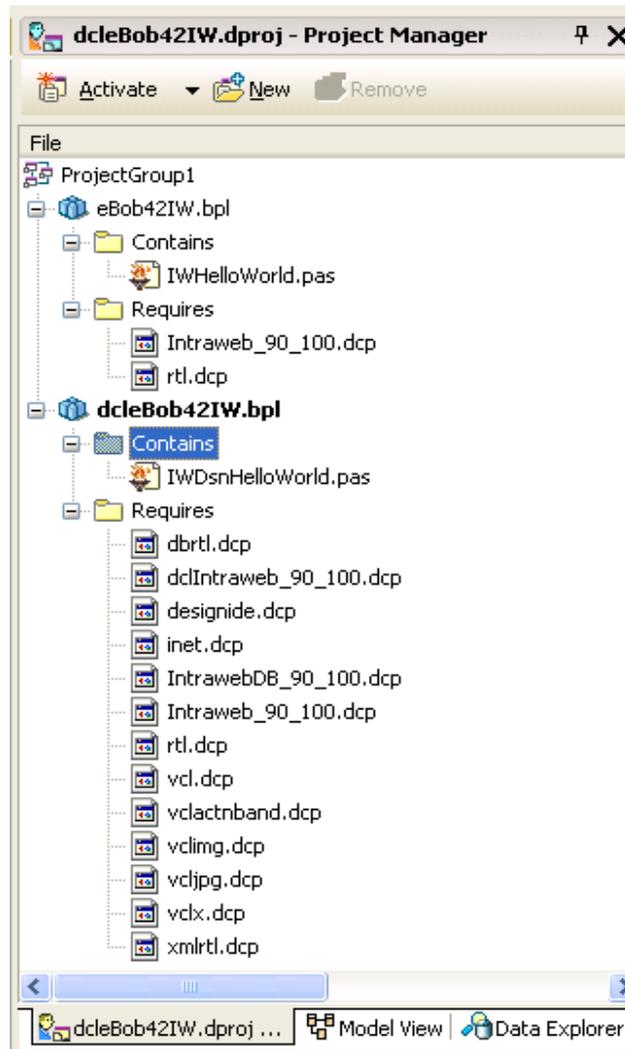
  procedure Register;

implementation
uses
  IWdSnPaintHandlers, IWBaseControl;

procedure Register;
begin
  RegisterComponents('IW Custom', [TIWHelloWorld])
end;

initialization
  IWRegisterPaintHandler('TIWHelloWorld', TIWPaintHandlerLabel);
end.
```

We can add both to their corresponding package, which results in the following overview for the run-time and design-time packages (see next page).



TIEuroComboBox

Existing IntraWeb components are easy to inherit from, like for example a TIWComboBox or TIWCustomComboBox component that gets extended by making sure it's pre-filled with items. This technique can be used to produce a TIWEuroCurrencies combobox (to select the 12 participating Euro currencies that can be converted to Euros and back), which is implemented as follows:

```

unit IWEuroComboBox;
interface
uses
  Classes, IWCompListBox, IWCompListBox32;

type
  TIWEuroComboBox32 = class(TIWCustomComboBox32)
  public
    constructor Create(AOwner: TComponent); override;
  published
    property ItemIndex;
    property Items;
  end;

```

```

TIWEuroComboBox = class(TIWCustomComboBox)
public
  constructor Create(AOwner: TComponent); override;
published
  property ItemIndex;
  property Items;
end;

implementation

constructor TIWEuroComboBox32.Create(AOwner: TComponent);
begin
  inherited;
  Items.Add('Euro'); // 1.0000
  Items.Add('ATS'); // 13.7603
  Items.Add('BEF'); // 40.3399
  Items.Add('DEM'); // 1.95583
  Items.Add('ESP'); // 166.386
  Items.Add('FIM'); // 5.94573
  Items.Add('FRF'); // 6.55957
  Items.Add('GRO'); // 340.750
  Items.Add('IEP'); // 0.787564
  Items.Add('ITL'); // 1936.27
  Items.Add('LUF'); // 40.3399
  Items.Add('NLG'); // 2.20371
  Items.Add('PTE'); // 200.482
  ItemIndex := 0 // Euro
end;

constructor TIWEuroComboBox.Create(AOwner: TComponent);
begin
  inherited;
  Items.Add('Euro'); // 1.0000
  Items.Add('ATS'); // 13.7603
  Items.Add('BEF'); // 40.3399
  Items.Add('DEM'); // 1.95583
  Items.Add('ESP'); // 166.386
  Items.Add('FIM'); // 5.94573
  Items.Add('FRF'); // 6.55957
  Items.Add('GRO'); // 340.750
  Items.Add('IEP'); // 0.787564
  Items.Add('ITL'); // 1936.27
  Items.Add('LUF'); // 40.3399
  Items.Add('NLG'); // 2.20371
  Items.Add('PTE'); // 200.482
  ItemIndex := 0 // Euro
end;

```

Again, we need to register the design-time paint handler, since we derived from `TIWCustomComboBox32` instead of - for example - the `TIWComboBox32` itself (which already has a paint handler defined, so we can use that one).

```

unit IWDsnEuroComboBox;
interface
uses
  Classes, IWEuroComboBox;

  procedure Register;

implementation
uses
  IWDsnPaintHandlers, IWBaseControl;

```

```

procedure Register;
begin
  RegisterComponents('IW Custom 32', [TIWEuroComboBox32]);
  RegisterComponents('IW Custom', [TIWEuroComboBox])
end;

initialization
  IWRegisterPaintHandler('TIWEuroComboBox32', TIWPaintHandlerComboBox32);
  IWRegisterPaintHandler('TIWEuroComboBox', TIWPaintHandlerComboBox)
end.

```

TIWRequiredEdit

As last example of an IntraWeb custom component, I want to derive a new TIWEdit control, this time from the regular TIWEdit (and not the TIWCustomEdit), and implement a warning message that will be shown when the user wants to leave the edit without entering a value (i.e. this will result in a so-called "required edit", and hence the name TIWRequiredEdit). The warning is implemented by adding an OnBlur event that will check the contents, and if empty will show the warning.

This custom IntraWeb component is implemented as follows:

```

unit IWRequiredEdit;
interface
uses
  Classes, IWCompEdit, IWScriptEvents;

type
  TIWRequiredEdit = class(TIWEdit)
  private
    FWarningMessage: String;
  protected
    procedure HookEvents(AScriptEvents: TIWScriptEvents); override;
  public
    constructor Create(AOwner: TComponent); override;
  published
    property WarningMessage: String
      read FWarningMessage write FWarningMessage;
  end;

implementation

constructor TIWRequiredEdit.Create(AOwner: TComponent);
begin
  inherited;
  FWarningMessage := 'Warning: a value is required.'
end;

procedure TIWRequiredEdit.HookEvents(AScriptEvents: TIWScriptEvents);
var
  Required: String;
begin
  inherited HookEvents(AScriptEvents);
  Required := 'if (this.value == '') { alert('' + FWarningMessage + '') }';
  AScriptEvents.HookEvent('onBlur', Required)
end;

end.

```

Note that this time we don't have to implement a paint handler, since the TIWRequiredEdit will simply use the already registered paint handler of its parent control the TIWEdit.

But you still want to keep the Register routine in a separate unit anyway, which is implemented as follows:

```
unit IWdsnRequiredEdit;  
interface  
uses  
  Classes, IWRequiredEdit;  
  
  procedure Register;  
  
implementation  
  
procedure Register;  
begin  
  RegisterComponents('IW Custom', [TIWRequiredEdit])  
end;  
  
end.
```

Installation and Usage

You can add the IWdsnXXX units to a package and install the components in the Delphi Tool Palette. When you place a component on an IntraWeb Application Form, only the run-time code will be linked with your application.

Summary

In this section, I've demonstrated how to create and deploy IntraWeb custom components using Delphi.

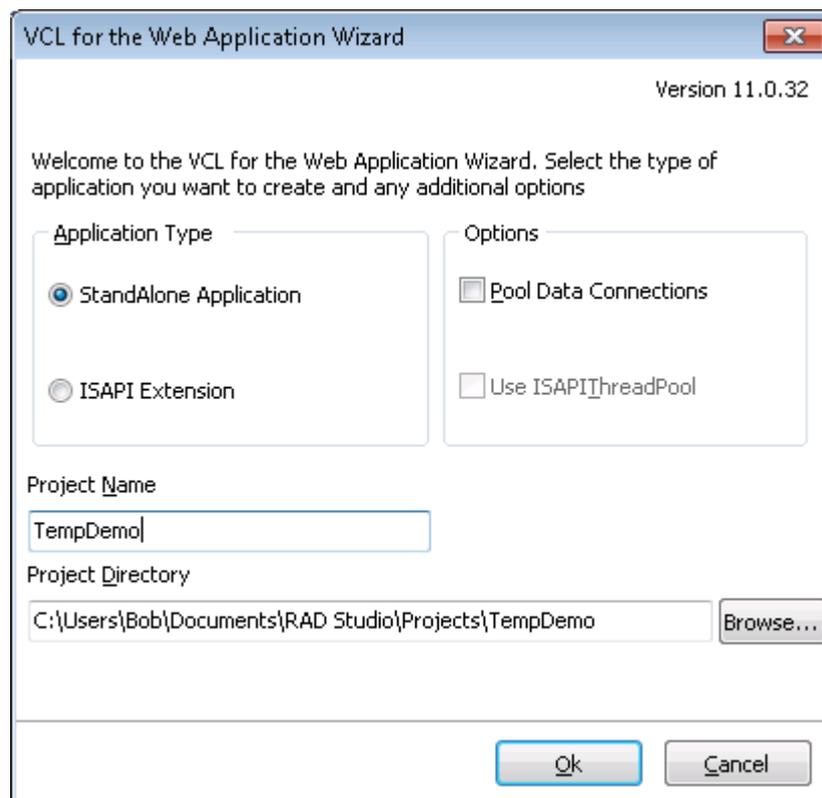
7. IntraWeb Testing Framework

Apart from writing web applications, IntraWeb 9 also includes support for unit testing which can be a great help in testing the output and result of your IntraWeb applications.

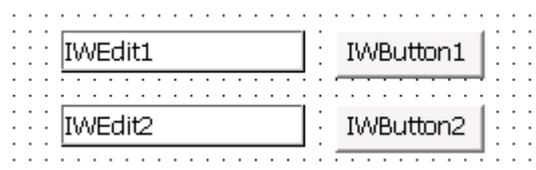
Sample Application

In order to demonstrate this unit testing behaviour, we should first create a simple IntraWeb application that does something – like a temperature conversion application – and then add the tests to it.

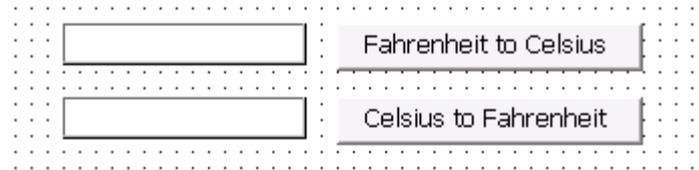
Using IntraWeb XI, use the VCL for the Web Application Wizard to create a new VCL for the Web application, with TempDemo as the project name.



Click OK to create the new VCL for the Web application. The new project will be called TempDemo, and contains a UserSessionUnit, ServerController unit as well as a Unit with the main form. Save the latter in MainForm.pas, and place two TIWEdits and two TIWButton control on it.



Rename the two TIWEdit controls to iwedFahrenheit and iwedCelsius, and rename the two TIWButton controls to iwbtnFahrenheit2Celsius and iwbtnCelsius2Fahrenheit (and don't forget to change their Caption property accordingly).



Note that the TIWEdits should be limited to numerical input only. This can be enforced in several ways, but is left as exercise for the reader. In the code below, I will use the built-in `StrToFloatDef` to convert the contents of the TIWEdit to a double, using 0 as default value if the contents is not a valid floating point value.

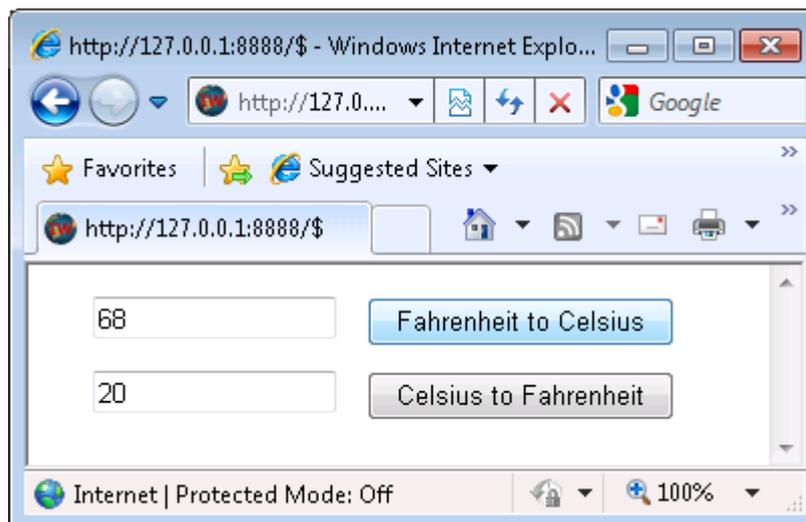
With this in mind, we can now implement the actual temperature conversion in the two `OnClick` events of the buttons, as follows:

```
procedure TIWForm4.iwbbtnCelsius2FahrenheitClick(Sender: TObject);  
begin  
    iwedFahrenheit.Text := FloatToStr(32 +  
        9 * StrToFloatDef(iwedCelsius.Text, 0) / 5);  
end;  
  
procedure TIWForm4.iwbbtnFahrenheit2CelsiusClick(Sender: TObject);  
begin  
    iwedCelsius.Text := FloatToStr(  
        5 * (StrToFloatDef(iwedFahrenheit.Text, 0)-32) / 9);  
end;
```

As basis for the code above, I've used the rule that degrees Fahrenheit are equal to 32 + the degrees in Celsius multiplied by 9/5. This code should perform the temperature conversion, but with code like this it's important to test the results in order to make sure I didn't accidentally forgot some brackets or replaced a multiplication by a division.

Manual Test

As a little test, I always convert 68 degrees Fahrenheit to 20 degrees Celsius (and back). And we can enter these values in the browser to verify the `OnClick` event handlers I've implemented.



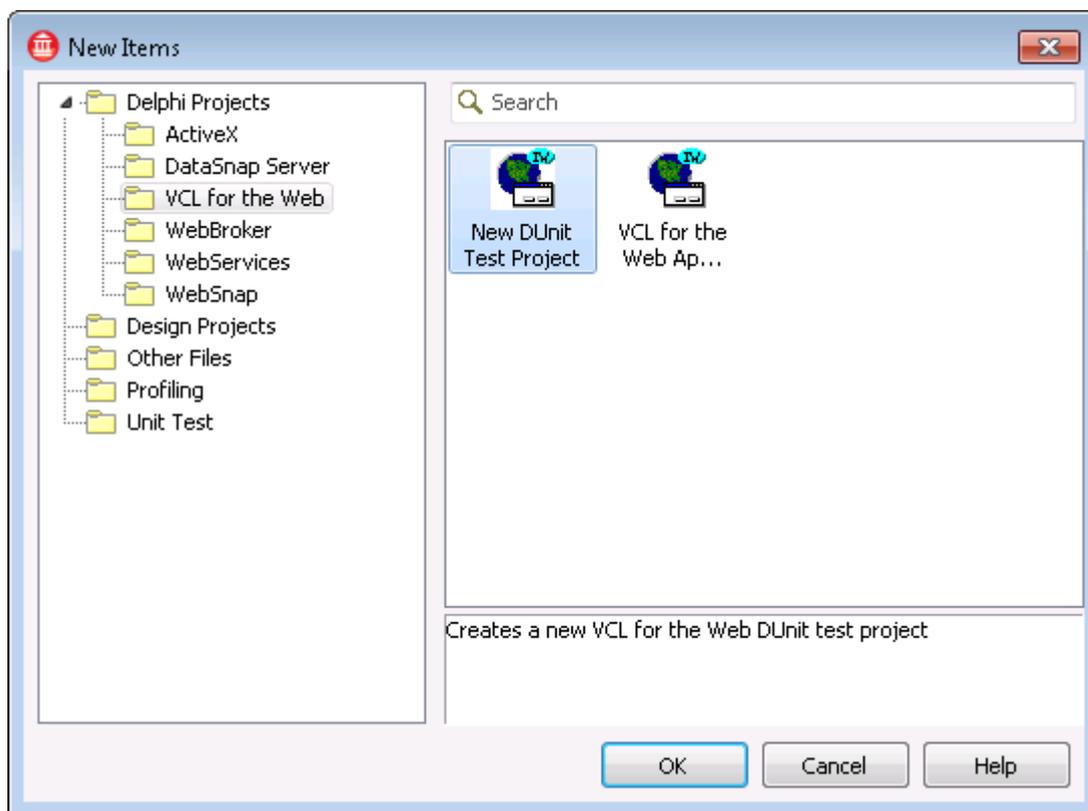
While this test is easy to perform, it would be a good idea to automate the test and to include some other test values as well (like 50:10), to ensure that I really correctly implemented the calculations behind the buttons.

For this and especially for more complex real-world examples, it's good to know that we can now use the IntraWeb Testing Framework to automate these tests, as I'll now demonstrate in the remainder of this section.

VCL for the Web Test Project

There are two ways to create a VCL for the Web Test Project. Either using File | New, or – in my view a better way – using the Add New Project option of the Project Manager. The former will create a new VCL for the Web DUnit Test Project, but without a reference to the VCL for the Web application you've just been working on. The latter will add a new VCL for the Web DUnit Test Project to the existing project group which makes it easier to allow the test project to work on the files from your original VCL for the Web project.

So, in the Project Manager, right-click on the ProjectGroup node and select Add New Project which will show the Object Repository again, where we can select the New DUnit Test Project icon from the VCL for the Web category:



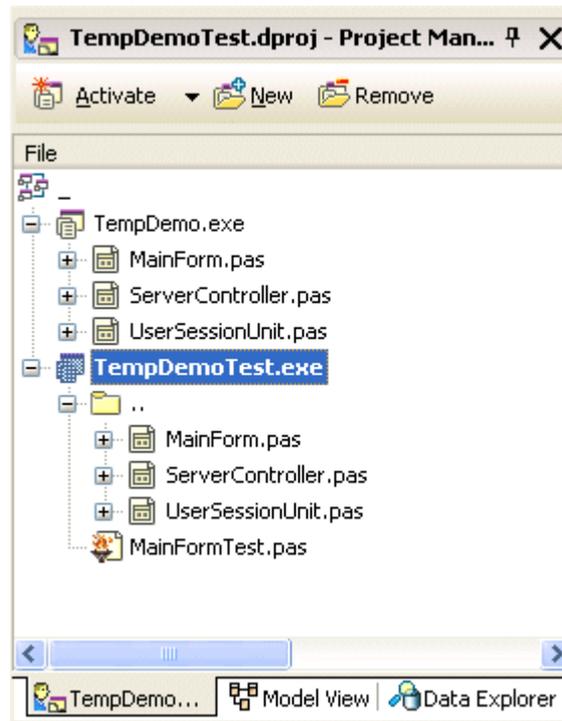
After you double-click on the New DUnit Test Project, you will not get a Wizard (like the normal DUnit test project in Delphi), but just a new project by default called Project1 with a test unit in Unit1.pas.

Personally, I prefer to keep my test project close to the actual project I'm testing. Like the normal Dunit Test Project Wizard in Delphi, my preferred location for the test project is the Test subdirectory of the actual project. Given that the TempDemo project was saved in my Projects\TempDemo directory, this means that the DUnit test project and test unit should be saved in the Projects\TempDemo\Test directory. I've saved the test project as TempDemoTest and the test unit as MainFormTest (in both cases I've used the original filename and added "Test" to it).

Sharing Main Form

The next step is an important one: in order to allow the test project to actually test the MainForm from the original project, we need to add unit MainForm to the test project. This can be done in the Project Manager again: right-click on the TempDemoTest project and select Add. In the dialog that follows, navigate to the parent directory (where the TempDemo project is located) and select the MainForm.pas unit.

Apart from the MainForm, however, we must also add the ServerController.pas and the UserSessionUnit.pas from the original IntraWeb TempDemo project to the TempDemoTest project. As a result, the Project Manager will have the following layout (note the shared files in the .. directory):



This will ensure that the DUnit test project shares the exact same units with the original TempDemo project, while the MainFormTest unit is the only new one, where the test code can be placed.

Writing Test Code

The next step involves writing test code for the MainForm. This can be done in the corresponding MainFormTest unit. First, add MainForm to the uses clause of MainFormTest, so the TIWForm type it can be used.

The unit MainFormTest contains the TIWTestCase class, which defines the Test method (only one, but we can add more if we want), as well as a SetUp and TearDown method, as follows:

```

type
  TIWTestCase1 = class(TTestCase)
  private
  protected
    procedure SetUp; override;
    procedure TearDown; override;
  published
    {$IFDEF CLR}[Test]{$ENDIF}
    procedure Test;
  end;

```

Note that we do not have to create an instance of our IntraWeb main form in the `SetUp` method (nor do we have to destroy it in the `TearDown` method). The `SetUp` and `TearDown` methods can be used for other things, like setting up a global lookup table or logfile. We do need to add `MainForm` to the `uses` clause of the `MainFormTest` unit, however.

Also note the `[Test]` attribute which is required in the .NET edition of the unit test. Although IntraWeb XE no longer supports .NET, this code is still present as witness of the previous support for both Win32 and .NET.

Inside the `Test` method, we can get our hands on an instance of the main form with the following code snippet which is part of the comments inside the `Test` method:

```
with NewSession do
try
  with MainForm as TIWFormXXX do
  begin
    ...
  end
finally
  Free
end
```

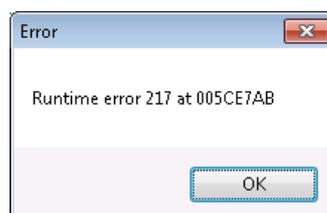
The call to `NewSession` will create a new session where we can access the `MainForm`. `TIWFormXXX` is the actual type of the main form we want to test, and is the type for which the `SetAsMainForm` method is called (in our case that's `TIWForm1`).

Since I prefer not to use (too many) `with` statements, the implementation of my `Test` method is slightly different. I'm using a `MyForm` variable to place the instance of `TIWForm1` in, and use the `MyForm` to access the controls (like `iwedFahrenheit` and `iwedCelsius`) and submit the buttons like `iwbtnFahrenheit2Celsius`.

The test is simple: I assign a text value of '60' to the `iwedFahrenheit.Text` property and then call the `Submit` method of the main form, passing the `iwbtnFahrenheit2Celsius` as argument. This will have the same effect as pressing the `iwbtnFahrenheit2Celsius` button in the browser, and as a result we can check the resulting value of the `iwedCelsius.Text` property, which should be 20.

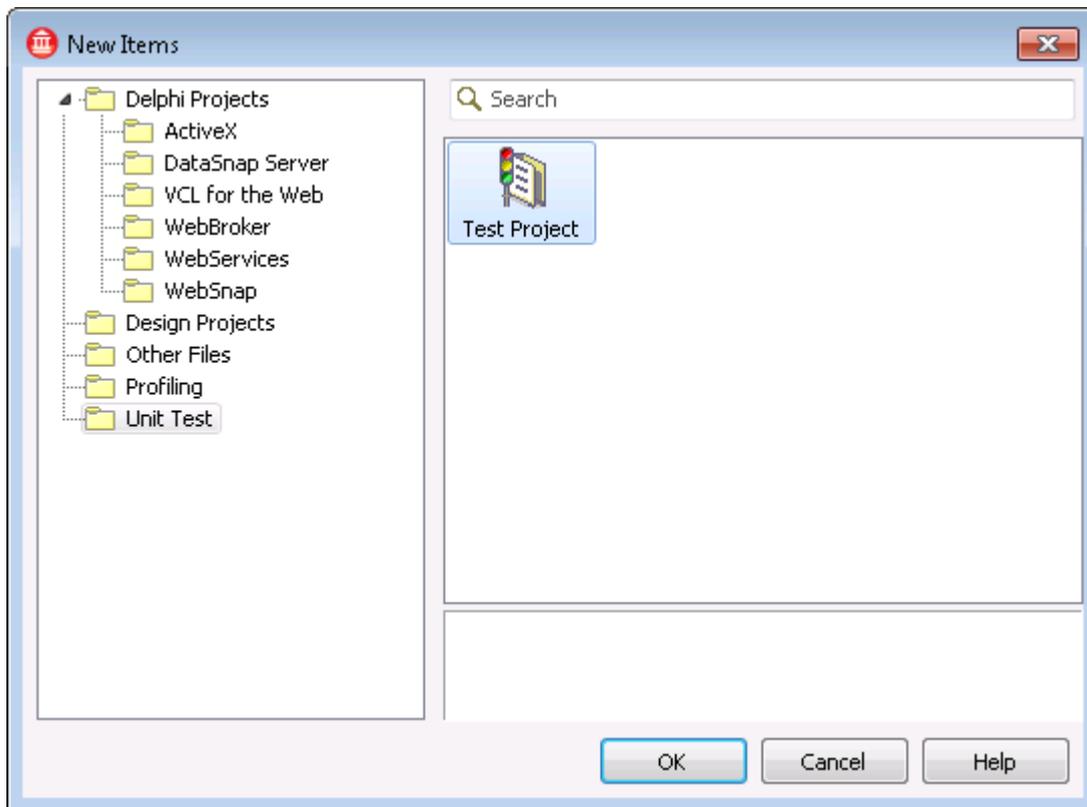
```
procedure TIWTestCase1.Test;
var
  MyForm: TIWForm1;
begin
  with NewSession do
  try
    MyForm := MainForm as TIWForm1;
    MyForm.iwedFahrenheit.Text := '68';
    MyForm.Submit(MyForm.iwbtnFahrenheit2Celsius);
    Check(MyForm.iwedCelsius.Text = '20',
      '68 Fahrenheit should be 20 Celsius');
  finally
    Free;
  end;
end;
```

If we compile and run the `TempDemoTest` application, we get a Runtime error 217, however.

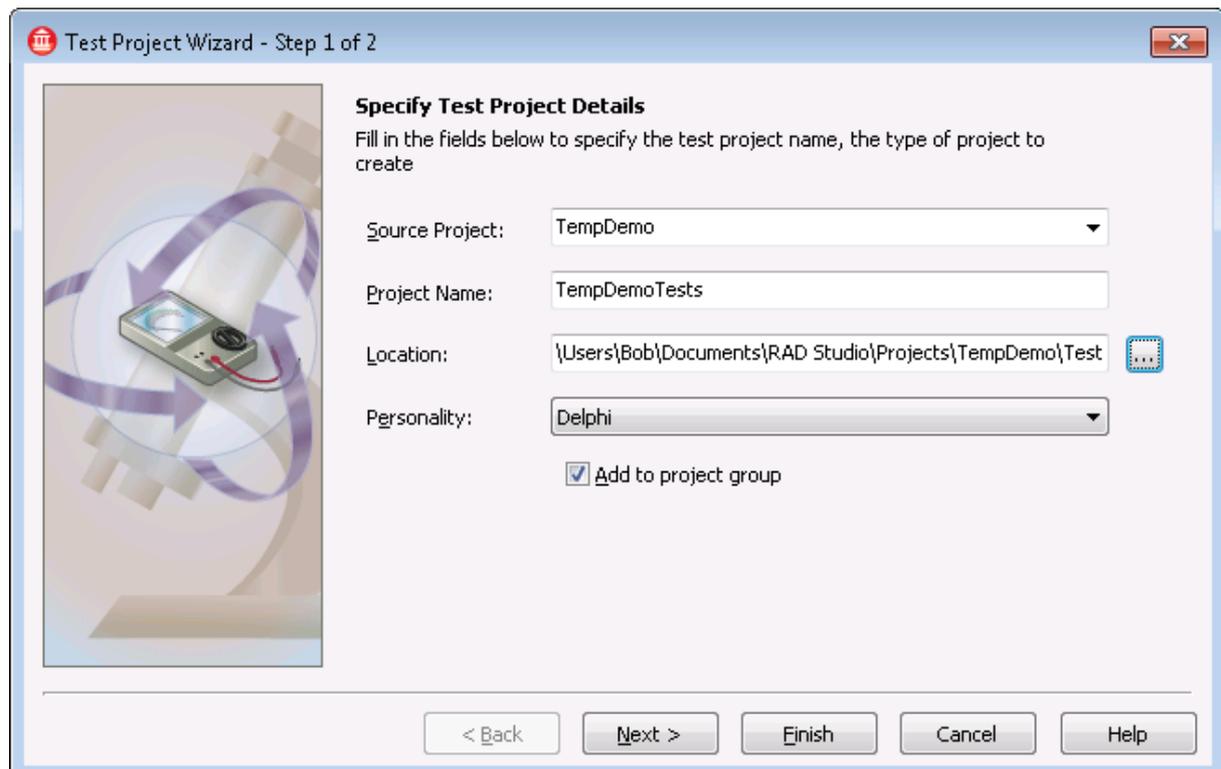


For some reason, there is a problem with the generated IntraWeb Test Project, and I've been unable to pinpoint the exact location of the problem.

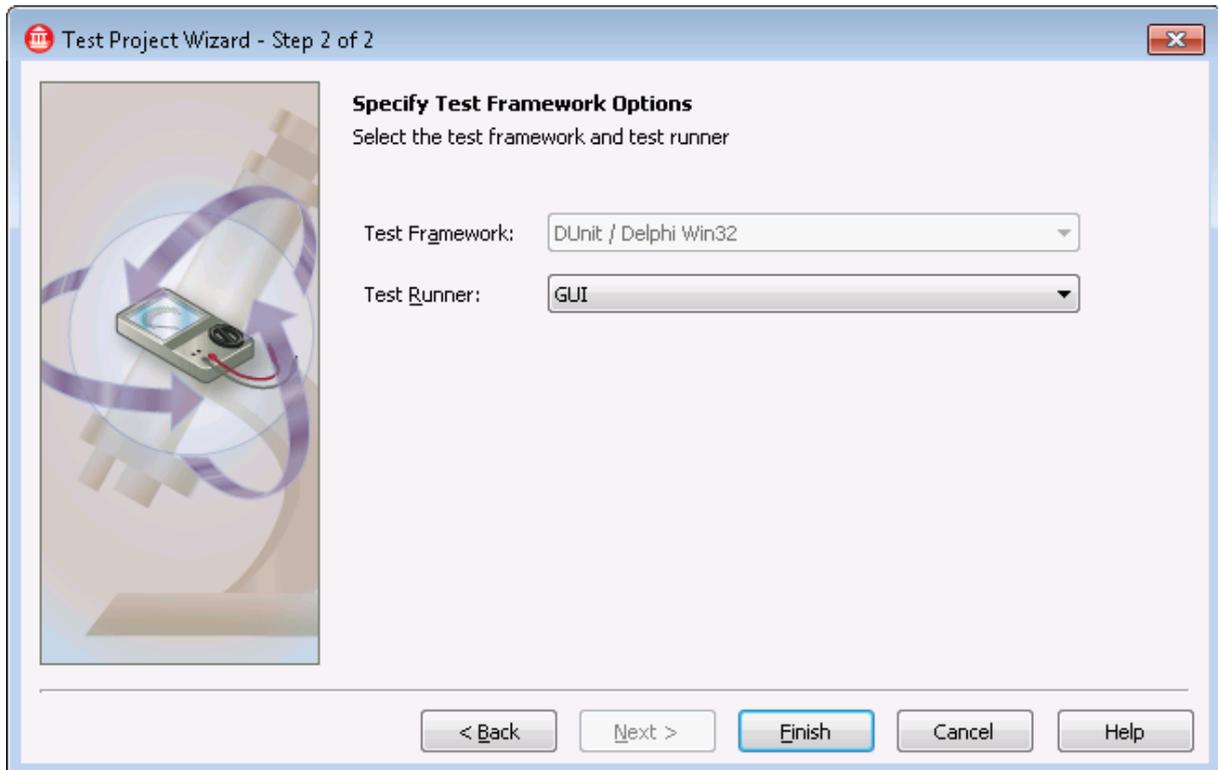
As a workaround, we can remove the IntraWeb DUnit test project, and just create a genuine Test Project from the Unit Test category in the Object Repository:



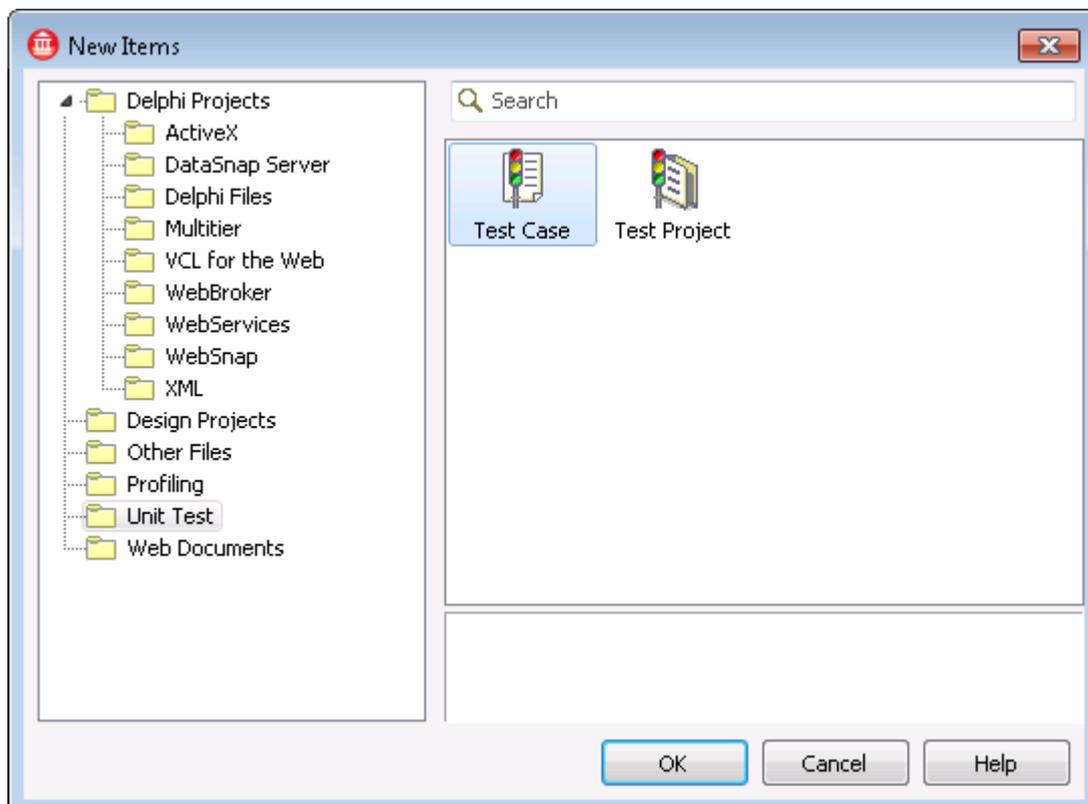
Select TempDemo as source project to generate the tests for.



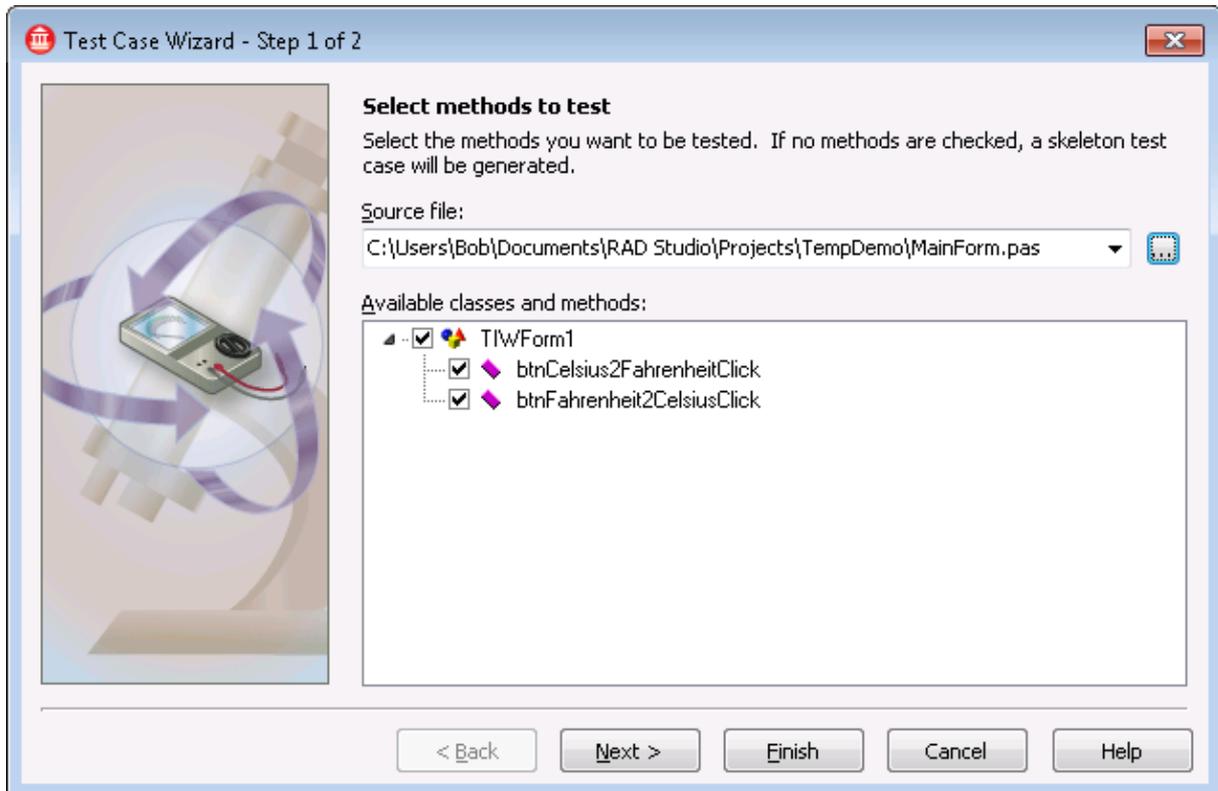
In the second step of the Wizard, we can select a GUI project (note that we can always switch to a CONSOLE test runner later).



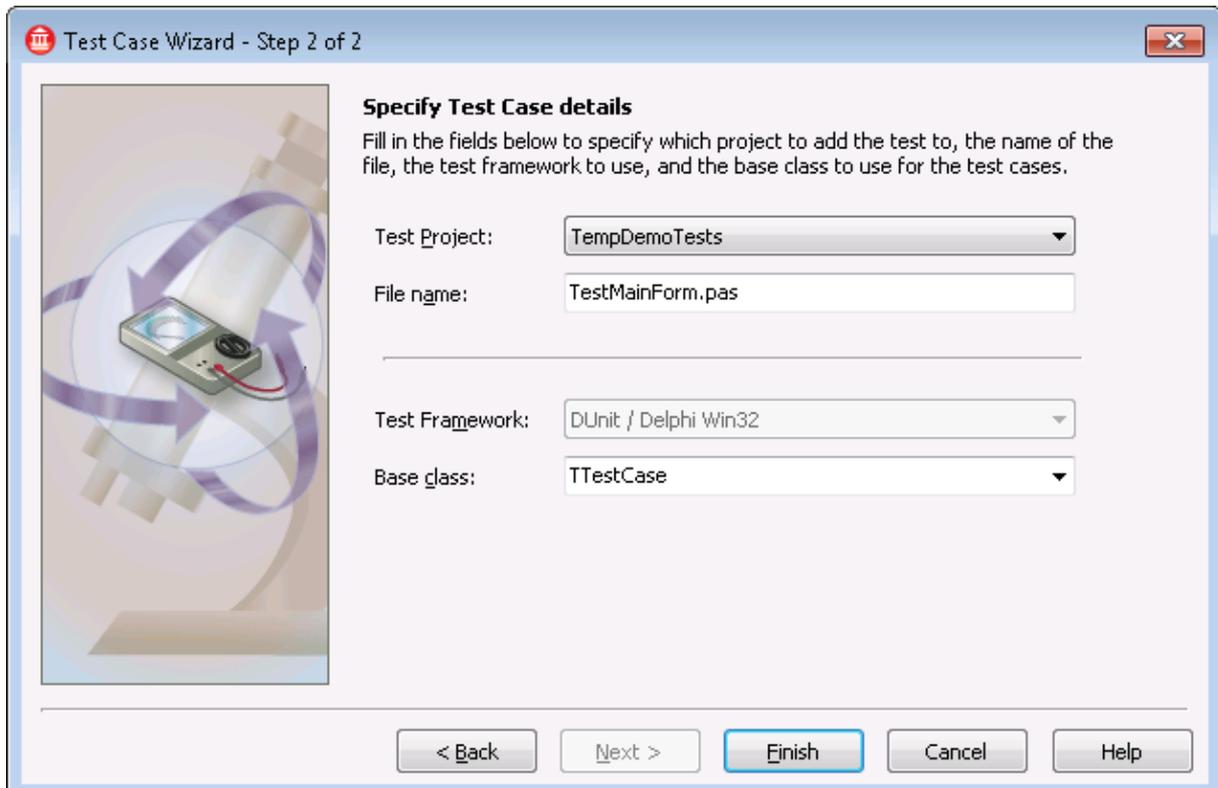
This will create an empty TempDemoTests project in the Test directory. We can add the MainFormTest.pas unit to it, as well as the MainForm itself, but it would be more instructive to see how we can add new tests for other IntraWeb forms as well. So add a Test Case from the Unit Test category in the Object Repository.



WE can now select the source files in the project directory. Make sure to select the MainForm.pas file, the one with the two events handlers btnCelsius2FahrenheitClick and Fahrenheit2CelsiusClick.



We can mark the methods we want to test, and then specify the target source file and base class:



We should now have a TestMainForm.pas unit added to the test project, as well as the MainForm.pas unit (but not the .dfm file).

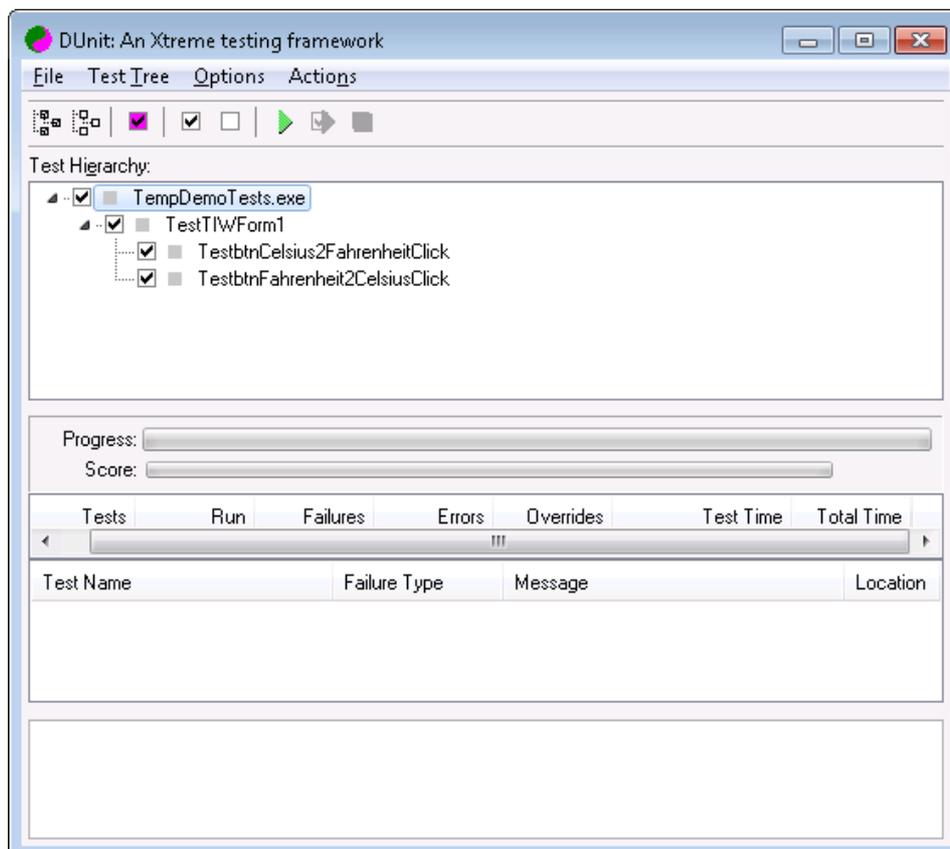
There are two test methods, which right now only consist of a call to the btnFahrenheit2CelsiusClick or btnCelsius2FahrenheitClick event handlers. We should modify them as follows:

```
procedure TestTIWForm1.TestbtnCelsius2FahrenheitClick;
begin
    FIWForm1.iwedFahrenheit.Text := '68';
    // TODO: Setup method call parameters
    FIWForm1.Submit(FIWForm1.btnFahrenheit2Celsius);
    // TODO: Validate method results
    Check(FIWForm1.iwedCelsius.Text = '20',
          '68 Fahrenheit should be 20 Celsius');
end;

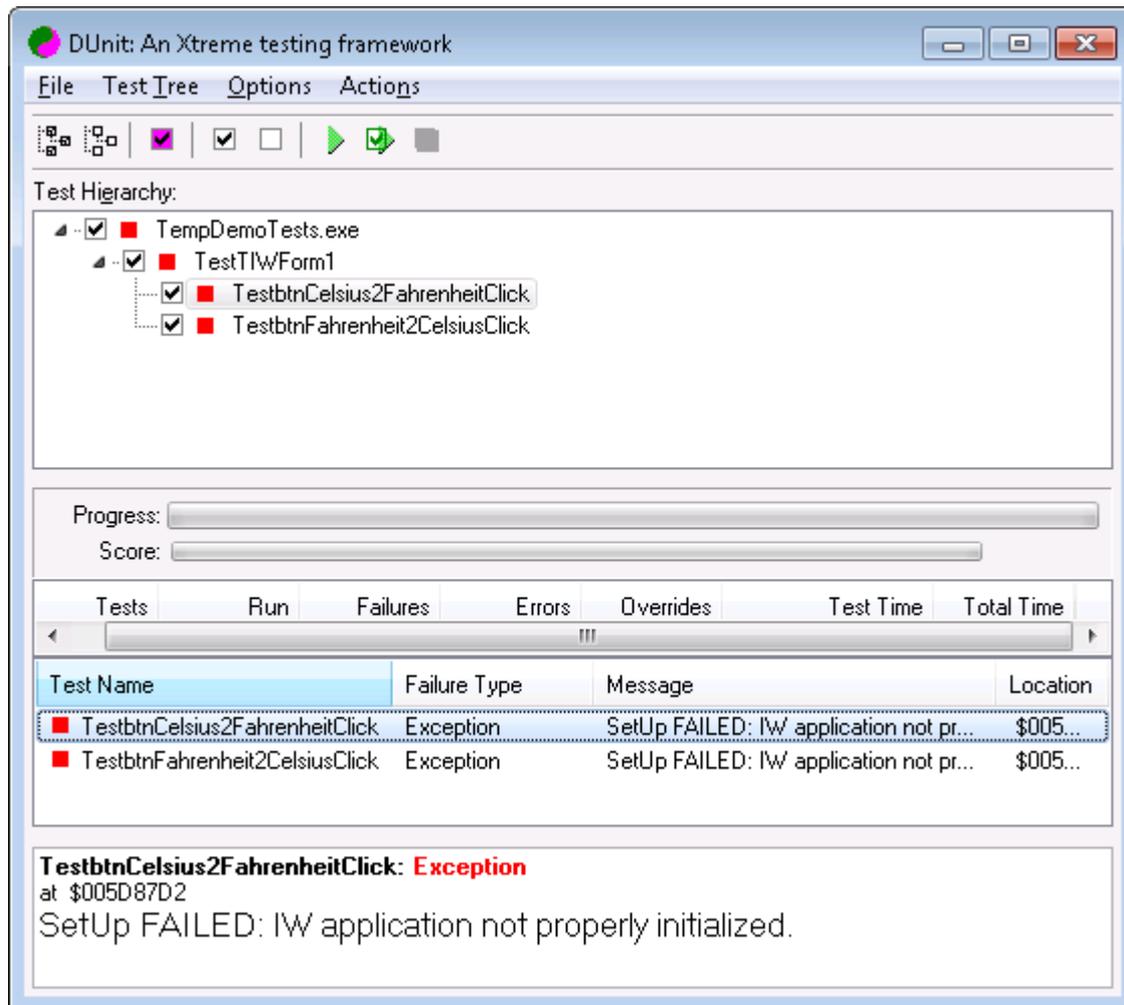
procedure TestTIWForm1.TestbtnFahrenheit2CelsiusClick;
begin
    FIWForm1.iwedCelsius.Text := '20';
    // TODO: Setup method call parameters
    FIWForm1.Submit(FIWForm1.btnCelsius2Fahrenheit);
    // TODO: Validate method results
    Check(FIWForm1.iwedFahrenheit.Text = '68',
          '20 Celsius should be 68 Fahrenheit');
end;
```

These two test methods verify that 68 degrees Fahrenheit gets converted to 20 degrees Celsius (first method) and back again (second method).

If we run this test project, we see the two tests appear in the DUnit framework:



However, running the tests will give an exception, which is caused by the fact that the IW Application itself is not property initialized.



Since our DUnit test project is a “generic” DUnit test project, and not one specific for IntraWeb, we forgot to correctly initialize the IntraWeb application. In order to fix that, we need to Open main project file, and add two units to the uses clause:

```
IWInit,  
IWGlobal,
```

And also add one line of code before the usual

```
if IsConsole then  
  with TextTestRunner.RunRegisteredTests do  
    Free  
  else  
    GUITestRunner.RunRegisteredTests;
```

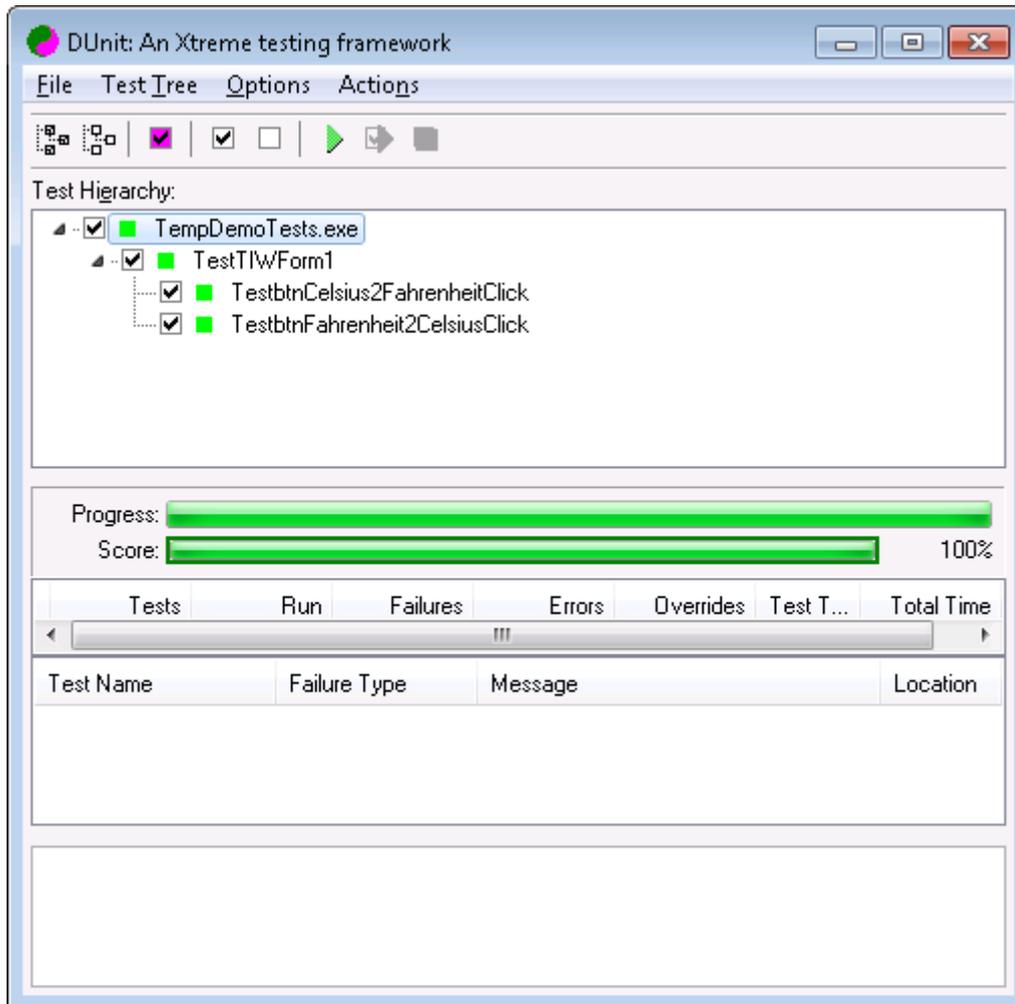
That's

```
GAppModeInit(Application);
```

The above single line of code will initialize the IntraWeb application for unit testing.

After that, we can compile and run the DUnit application framework again.

For the two tests that we've implemented, the result will be a green light (meaning a successful test). We can click on the green arrow or select the *Test Tree | Run Selected Test (F8)* option to run all checked tests – in this case just the one test.



If a test fails, we'll see a pink/purple colour as well as the text that was specified as second argument of the Check method. So apart from the test success / failure, you'll also get more details about the actual failure.

As an additional benefit: once a test has been implemented, it will never go away. And you can re-run the same test later (especially handy if you decide to change the implementation of the OnClick methods of the button in order to archive a better performance or perhaps for some other reason). Existing tests can be re-run just by running the DUnit test application again. Compared to running the original application in the browser and entering all test values and clicking on all buttons again, this is a very convenient way to ensure there is no regression or new bugs that have crept into the application.

More Tests

If you want to add more tests – which is always a good idea – you only need to add a new public method to the TIWTestCase class, and implement it.

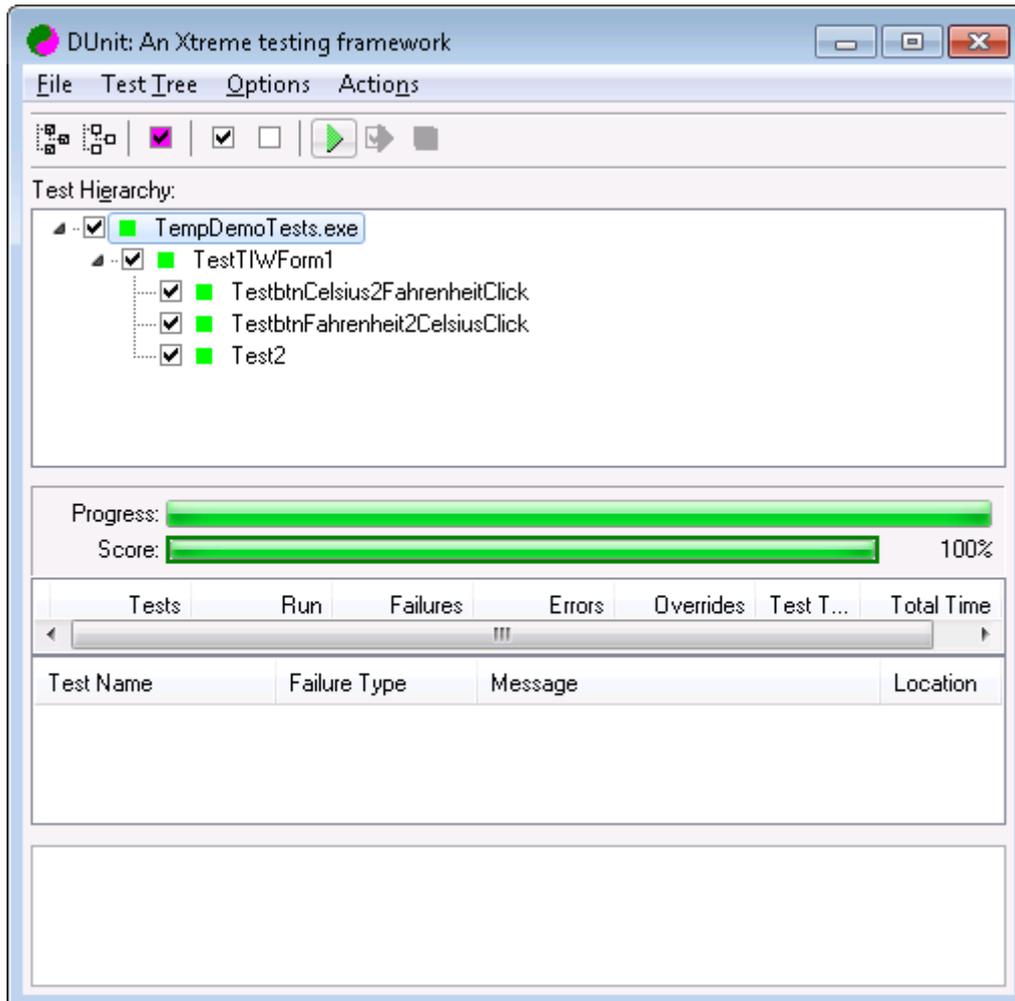
For example, to ensure that 10 degrees Celsius converts to 50 degrees Fahrenheit, I could add a Test2 method with the following implementation:

```

procedure TIWTestCase1.Test2;
begin
  FIWForm1.iwedCelsius.Text := '10';
  // TODO: Setup method call parameters
  FIWForm1.Submit(FIWForm1.btnCelsius2Fahrenheit);
  // TODO: Validate method results
  Check(FIWForm1.iwedFahrenheit.Text = '50',
        '10 Celsius should be 50 Fahrenheit');
end;

```

And this also results in green lights for a successful test:



I leave it as exercise for the reader to implement a test which will fail (hint: you can either make sure the OnClick event of the original application is incorrect, or add a check in the test form which expects an incorrect result).

I TestSuite

Note that all our public test methods are automatically picked up and displayed in the DUnit test application. This is done behind the scenes by the single line of code in the initialization section of the MainFormTest unit:

```

initialization
  RegisterTest(TestTIWForm1.Suite);
end.

```

It is, however, also possible to explicitly register the Test methods that you want to run, using a function that returns the ITestSuite interface, as follows:

```
function Suite: ITestSuite;  
begin  
    Result := TTestSuite.Create('Temperature Conversion Tests');  
    Result.AddTest(TIWTestCase1.Create('Test'));  
    Result.AddTest(TIWTestCase1.Create('Test2'));  
    ...  
end;  
  
initialization  
    RegisterTest('Temp Tests', {TestTIWForm1.} Suite);  
end.
```

Note that instead of calling `TIWTestCase1.Suite` (the implicit `ITestSuite` that is generated by the `TIWTestCase1` class itself), we then have to call the global function `Suite` which contains the information as well as the list of Test methods we want to display inside the DUnit test application. Using comments or `IFDEFs` you can then easily enable or disable tests.

Summary

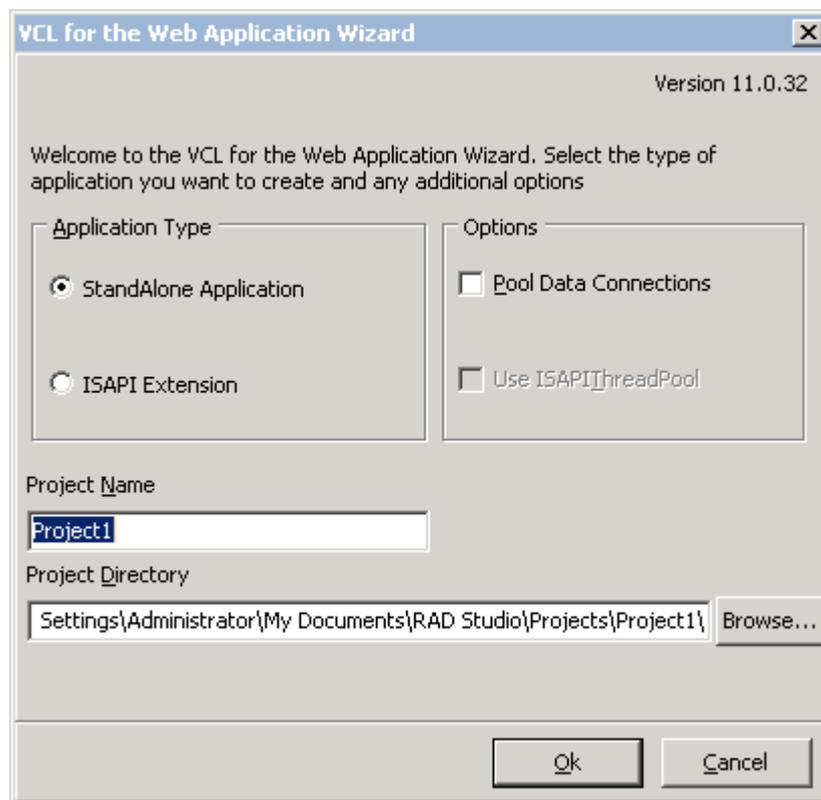
In this section I've described and demonstrated how we can use the VCL for the Web DUnit test framework to add tests to our VCL for the Web applications. An important benefit of using a separate test application is that the original VCL for the Web application remains unchanged, so you do not introduce strange behaviour (like the fact that a bug no longer exists after you add test or debug code). The biggest benefit of using the DUnit test framework, however, is the fact that you only have to write the test once, and can run and re-run the test several times, including months or years later, in order to verify that a certain piece of functionality is still working as it should be.

8. IntraWeb Deployment

So far, all examples in this VCL for the Web / IntraWeb courseware manual have started the web application as a stand-alone executable. Which is the easiest way to start, and also the best way to debug your web application. However, when it comes to deployment, there may be cases when you need or want to deploy the application in a different way.

Project Targets

When creating a new VCL for the Web application, we have a choice between different targets: StandAlone Application or ISAPI Extension.



We can easily turn the standalone application into a Windows Service, as discussed in the first section of this courseware manual.

StandAlone Application

The StandAlone Application needs no further explanation, and has been used for most of the examples in this courseware manual. There main advantage of the StandAlone Application is that fact that you can run and debug it from inside the IDE. Breakpoints and easy of debugging is a great way to allow you to find bugs or problems in your source code.

However, when it comes to deployment of the IntraWeb application, your StandAlone Application may need some additional configuration on the server machine which may not always be possible. Specifically, you need to ensure that the firewall allows access to the port number which is used by your StandAlone Application (a random port number when using the evaluation version of IntraWeb). This is no problem on your development machine, but may be an issue on the deployment machine, especially if the deployment machine is managed by an ISP or hosting provider (where you do not

have 100% control over the web server machine – most ISPs are not happy opening up additional ports of their firewall, for obvious reasons). As a result, StandAlone Applications are best to deploy in environments where you have total control over the web server yourself.

Service Application

We can turn the StandAlone Application into a Service Application by passing False to the call of TIWStart.Execute in the main project file.

```
TIWStart.Execute(False);  
end.
```

With this setting, the application is not forced to start in GUI mode, and we can pass the /install command line option to actually install the service:

```
IWDemo.exe /install
```

Once the IntraWeb Service application has been installed, we can start it using the Services console of Windows. Note that for a Service Application, it's very important to make sure that the AppName property of the ServerController is set to a unique name. Also, the IntraWeb Service application uses the port number as specified in the Port property of the ServerController to communicate with the outside world, so like the StandAlone Application you must ensure that the firewall allows incoming connections to be made at this port number. Combined with the fact that the service must be installed on the web server, this may mean that ISPs are generally unwilling to host IntraWeb Service applications – but they can be deployed on a web server which is under your own control, of course. A benefit of a Windows Service applications compared to the IntraWeb StandAlone application is the fact that the Windows Service can get a start more of Automatic, which means that it will be started automatically (when Windows boots) and you do not need a user to be logged on at the web server machine (whereas the StandAlone Application of course needs a Windows session, and hence someone to be logged on at that console).

ISAPI Extension

For deployment on a web server machine which is under the control of an ISP or other external party, the ISAPI extension is the most suited format. For an ISAPI Extension, the Port property of the ServerController is ignored, since the ISAPI application will be loaded by Internet Information Service (IIS) which defines its own ports to communicate with the outside world. By default, this is port 80 for normal and port 443 for secure connections.

The ISAPI application must be deployed in a virtual directory on the web server which has the Execute rights enabled (also note that you may have to explicitly enable ISAPI Extensions on the web server machine, for example on Windows Server 2003 with IIS6 or Windows Server 2008 with IIS7).

Note that the ISAPI Extension is only available for IntraWeb XI Ultimate Edition!

Multiple Project Targets

Although it's possible to migrate the Delphi project from one type to another (for example from the StandAlone Application to an ISAPI Extension project), this is not a solution I would recommend. If only because once you have an ISAPI project, it may be easier to deploy, but harder to debug.

Personally, I prefer to have the best of both world: being able to debug the StandAlone application, install and run the Service application (when needed), or deploy the ISAPI application on an external web server machine. This is only possible if we make sure that all project targets share the exact same source files.

Windows Server 2003

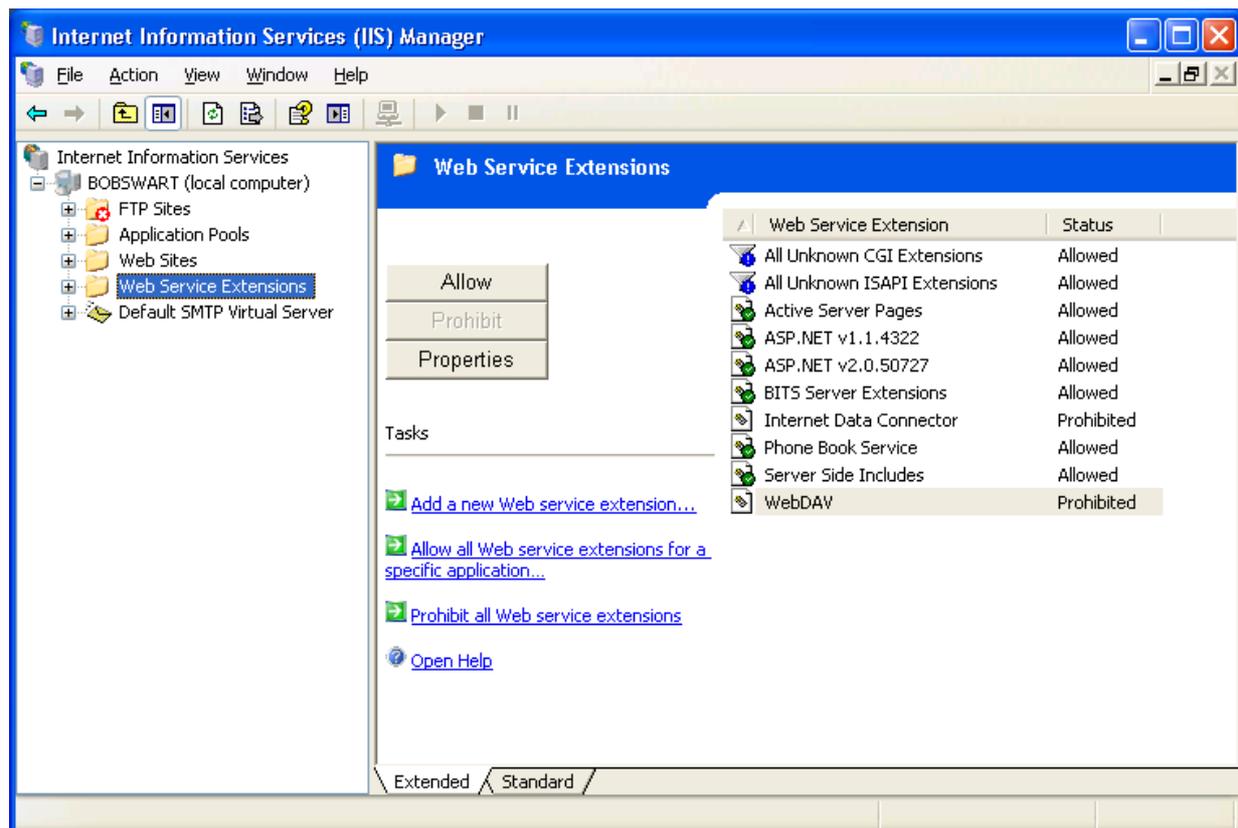
For real-world deployment on Windows Server 2003, using Microsoft Internet Information Services (IIS) version 6, you can following the steps described here. For deployment on Windows Server 2008 and IIS 7.1, please refer to the next section which includes detailed steps for that situation.

Enabling ISAPI / CGI

Internet Information Services (IIS) version 6 on Windows Server 2003 explicitly disables support for CGI or ISAPI extensions, and has to be configured to handle these application types. In order to configure IIS, we need to start the Internet Information Services (IIS) Manager, either directly from the Administrative Tools, or from within the Computer Management console. The later can be started with a right-click on the My Computer icon, but we'll use the former (since this only shows the IIS Manager screen in the screenshots).

In the Internet Information Services (IIS) Manager dialog, open up the local computer node, and then select the Web Service Extensions node. This shows the known Web Service Extensions such as CGI, ISAPI, Active Server Pages, ASP.NET 1.1, ASP.NET 2.0, etc.

We need to select the options for All Unknown CGI Extensions and All Unknown ISAPI Extensions, and explicitly change the status to Allowed. Both these extensions will be disabled in IIS6 on Windows Server 2003 by default, even in the Web Edition of Windows Server 2003, so it's important to explicitly allow them.

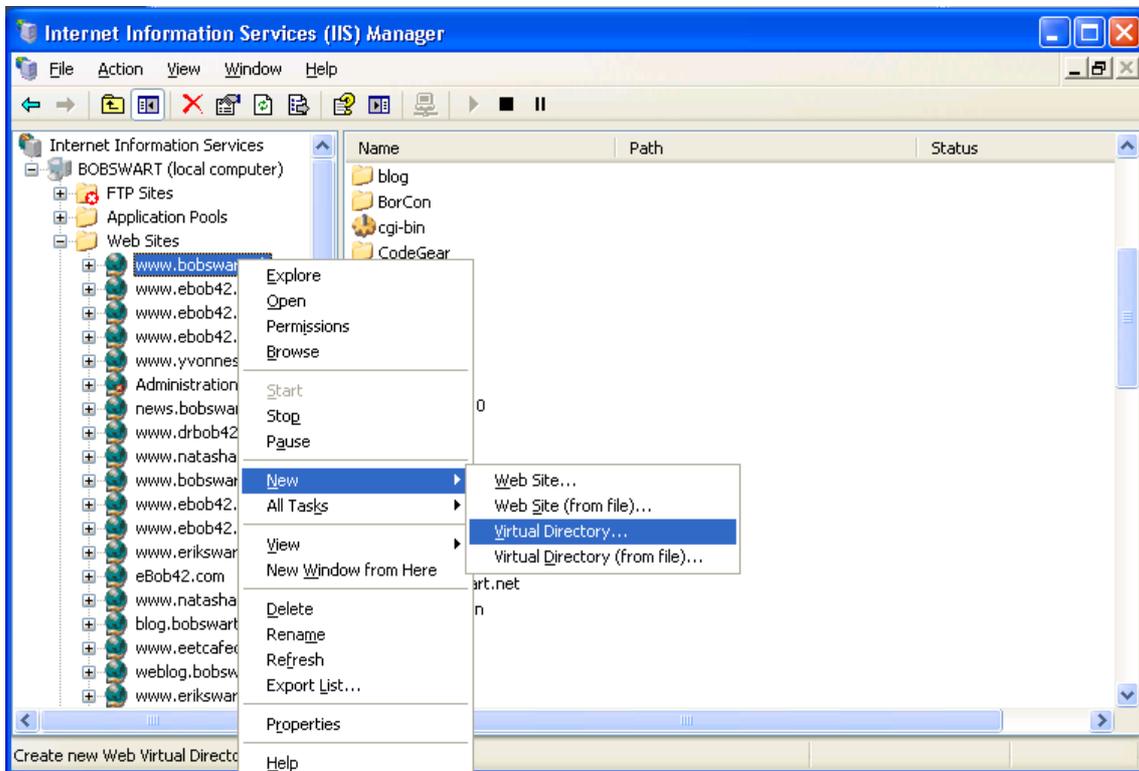


If you know in advance that you only need to deploy CGI executables – or ISAPI extensions – you may decide to allow only the required extension here: either All Unknown CGI Extensions or All Unknown ISAPI Extensions.

Once the CGI and/or ISAPI Extensions are enabled, we can create a virtual directory where the CGI executable or ISAPI dynamic link library can be placed for deployment.

Virtual Directory

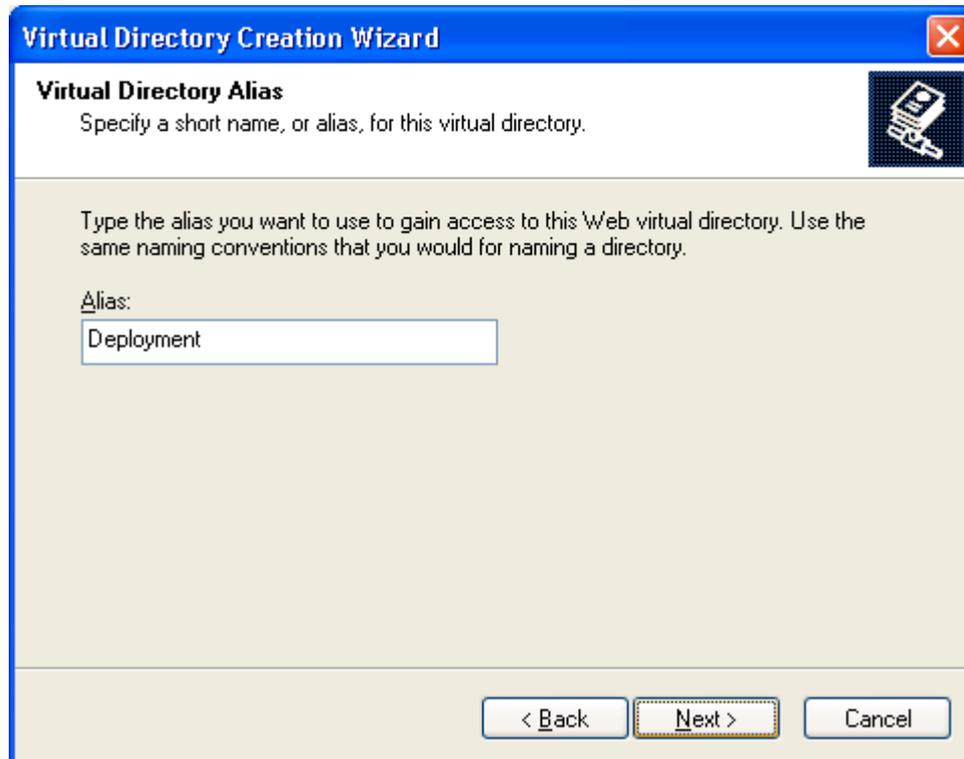
On Windows 2003, we need to deploy the CGI or ISAPI application in a virtual directory that has the execute rights enabled. In the previous screenshot, open the Web Sites node and select the web site for which you want to create the virtual directory. Then, right-click on the web site node, and do *New | Virtual Directory*.



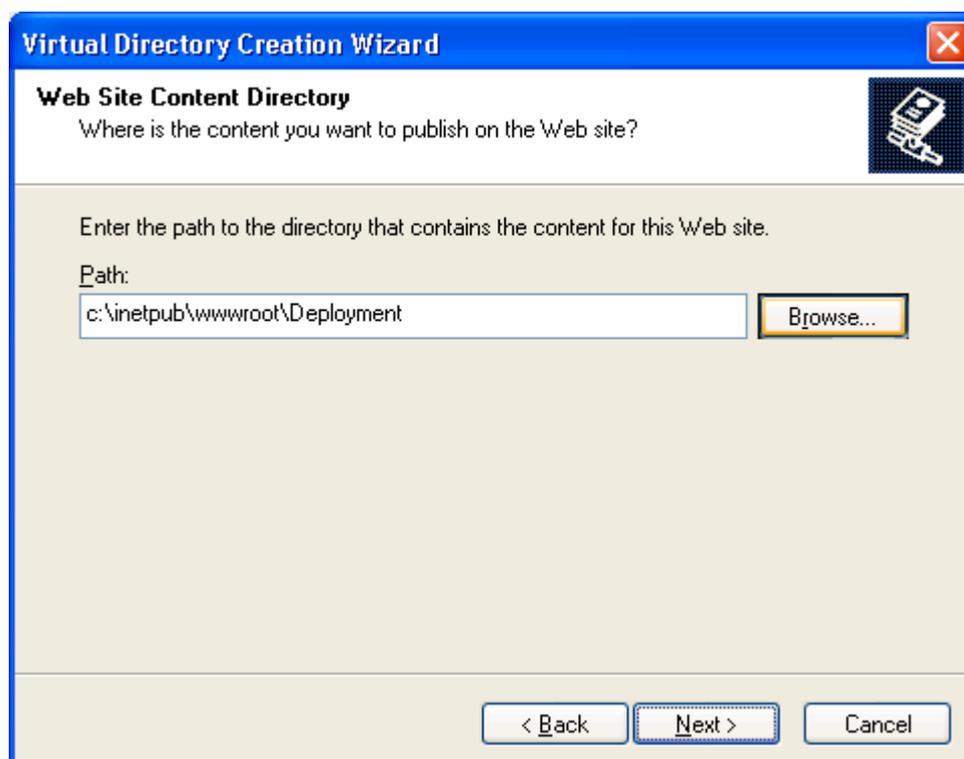
This will produce a Virtual Directory Creation Wizard that takes you through the steps to create and configure a virtual directory.



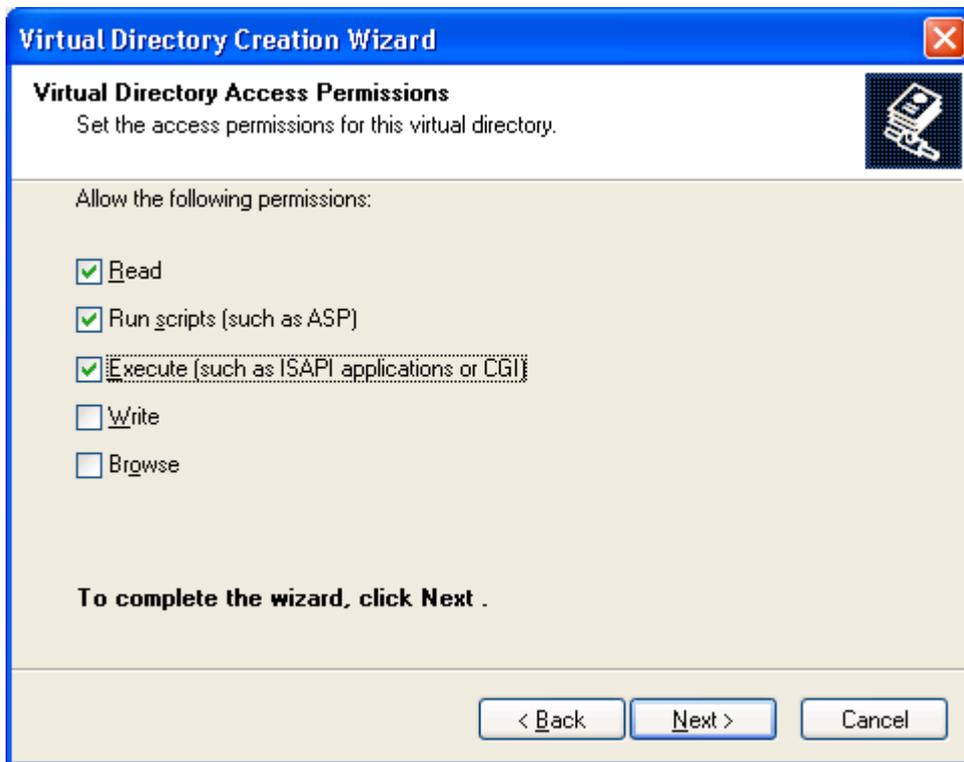
You first have to specify the alias for the virtual directory. Specify Deployment for the example from this courseware manual (feel free to use your own name for the virtual directory, of course, but the dialogs will use Deployment as name).



The next step is the location for the virtual directory, for which I usually pick a directory like `c:\inetpub\wwwroot\Deployment`, or a subdirectory in the root of the specific web site I'm creating this virtual directory for. Note that the directory must exist, otherwise you will get an error message telling you it doesn't exist.



The last step in the Virtual Directory Creation Wizard dialog involves setting the permissions for the virtual directory. For ASP.NET web services, you only need to enable Read and Run scripts (such as ASP) and no Execute, Write or Browse rights. But for CGI executables and/or ISAPI DLLs, you need to check the Execute rights option.



When the Execute option has been set, you can continue and finish the Virtual Directory Creation Wizard.

Once the virtual directory is created, you can right-click on the virtual directory and select properties to continue with the configuration, allowing you to set the default content pages (by default set to index.htm, index.html and Default.aspx for example), but also the Directory Security settings, and the virtual directory options themselves. The virtual directory will contain Application Settings, in our case with Application name Deployment, using the default application pool. It's a good idea to use a special application pool for certain virtual directories and applications, so they do not get in the way of other applications.

You can use the Application Pools node (just above the Web Sites node) to configure existing application pools and/or add new ones for your virtual directories and applications.

You can now place your CGI, ISAPI or ASP.NET projects in the new virtual directory. For the WebServiceCGI.exe placed in the Deployment directory, the URL to call this will be <http://localhost/Deployment/WebServiceCGI.exe> where you can replace localhost by the IP-address or – even better – the DNS name of the machine. Assuming this was my own web server, and I just created the virtual directory Deployment on the www.bobswart.nl site, then the result would be <http://www.bobswart.nl/Deployment/WebServiceCGI.exe>.

In a similar way, the WebServiceISAPI.dll can be called from the same virtual directory with the URL <http://localhost/Deployment/WebServiceISAPI.dll> or with your real name in place of the localhost.

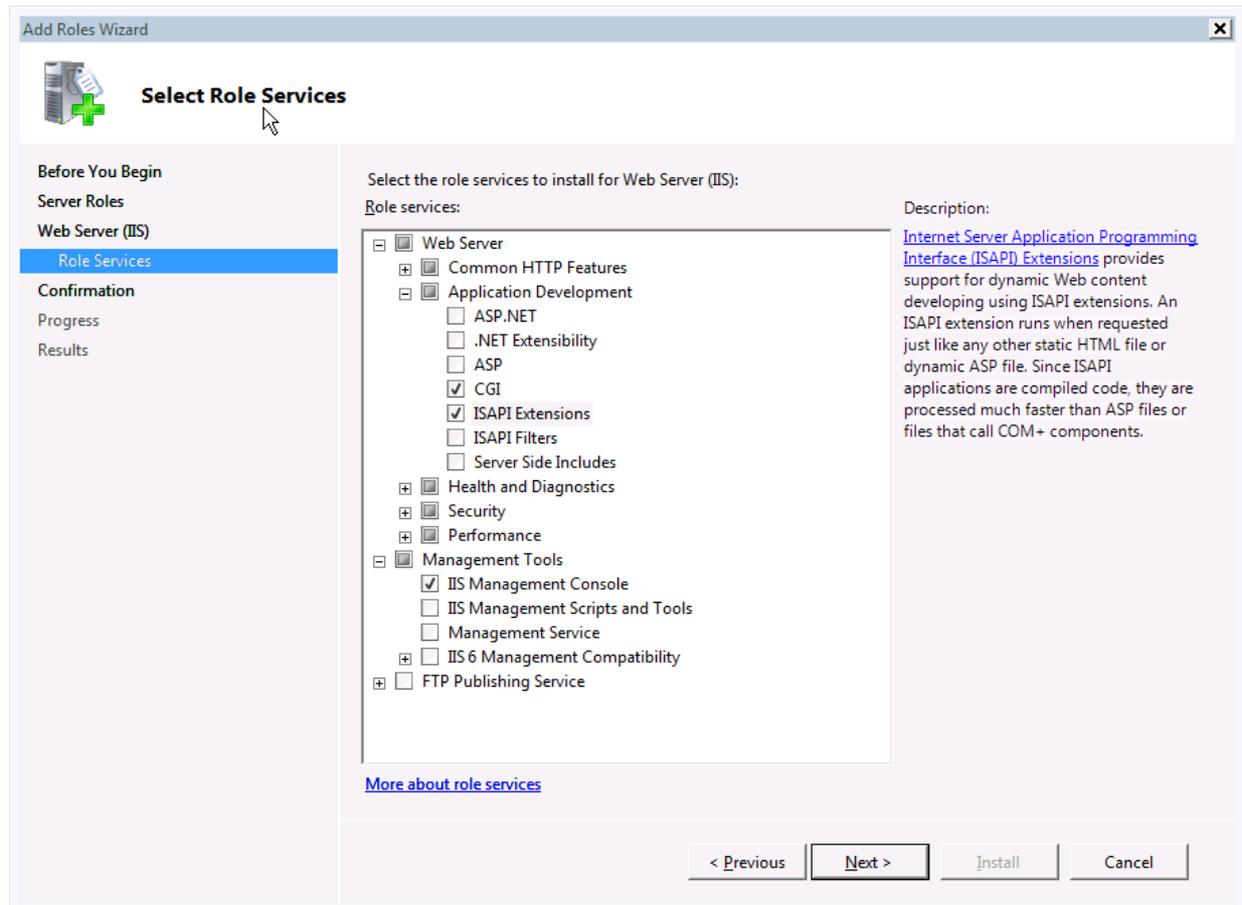
Note that if you do not have your own web server machine, there is a good chance you need help from your ISP in order to create the virtual directory.

Deployment on Windows Server 2008 and IIS7

Windows Server 2008 comes with Internet Information Server (IIS) version 7, which looks and feel different (and works different) compared to IIS6 on Windows Server 2003.

First of all, when working with Windows Server 2008, you have to make sure to install Internet Information Services, and the "CGI" and/or "ISAPI Extensions" Application Development Features.

Note that you may only need to select one of these if you decide to deploy only CGI or ISAPI extensions instead of both types!

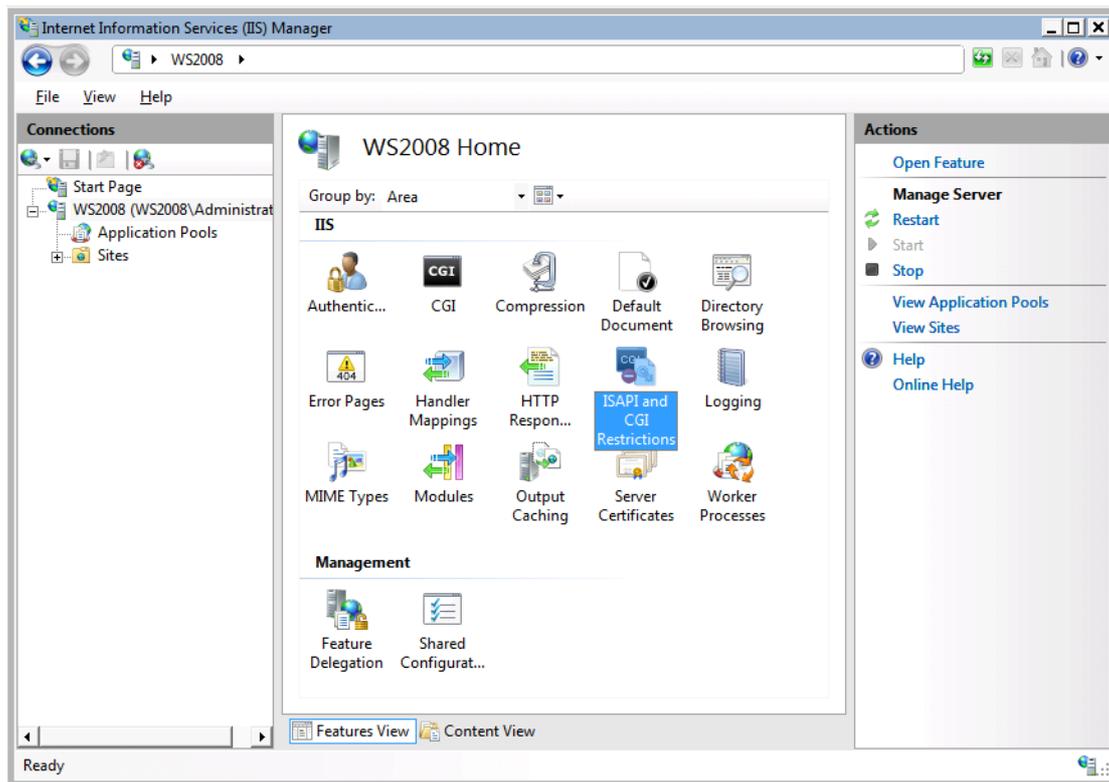


In the above screenshot, both CGI and ISAPI Extensions have been selected. But in practice, you may already have decided to deploy only CGI executables, or only ISAPI DLLs, in which case you only have to configure the CGI or ISAPI Extension options of course.

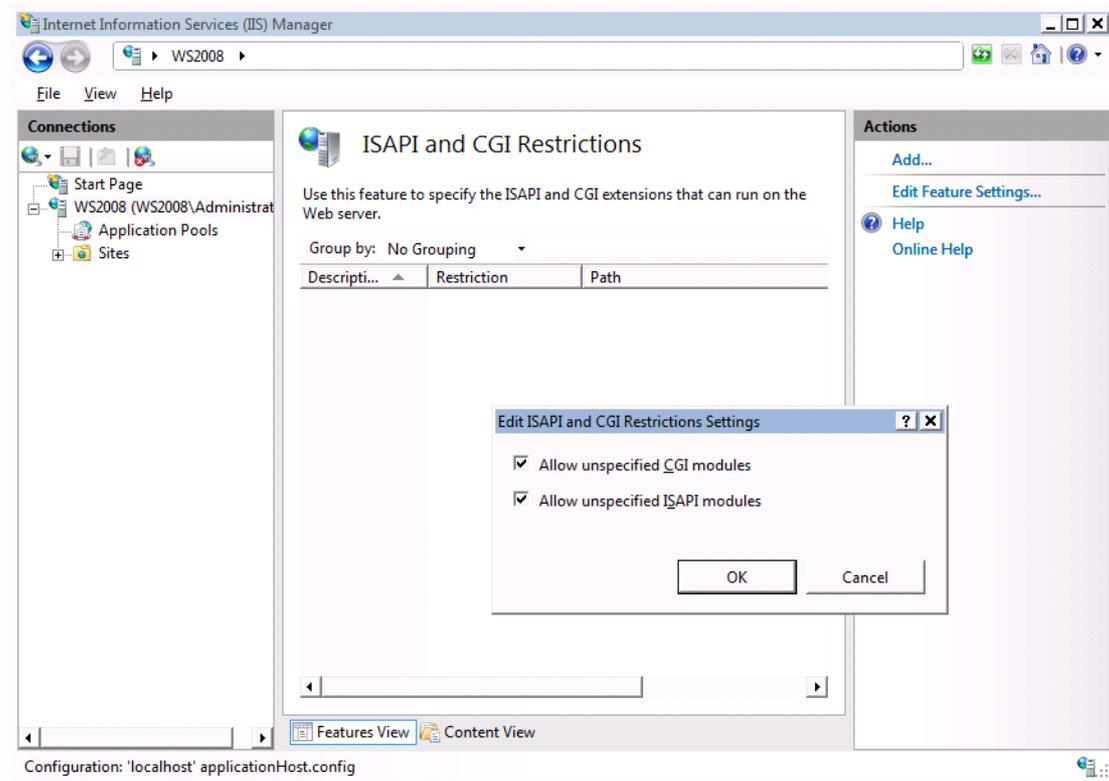
Once IIS7 is installed correctly, and you can view <http://localhost> in a browser, it's time to configure IIS and add the virtual directory.

From the Administrative Tools in the Windows' Start menu, select the Internet Information Services (IIS) Manager. Alternately, right-click on the My Computer icon and select Manage. Then, inside the Server Manager dialog, open the Roles node and select the Web Server (IIS) node. The subnode Internet Information Services (IIS) Manager will show you the IIS Manager as well.

In the windows that follows, select your machine (WS2008 in my case), and then in the middle of the screen select the "ISAPI and CGI Restrictions" icon (see screenshot on the next page).

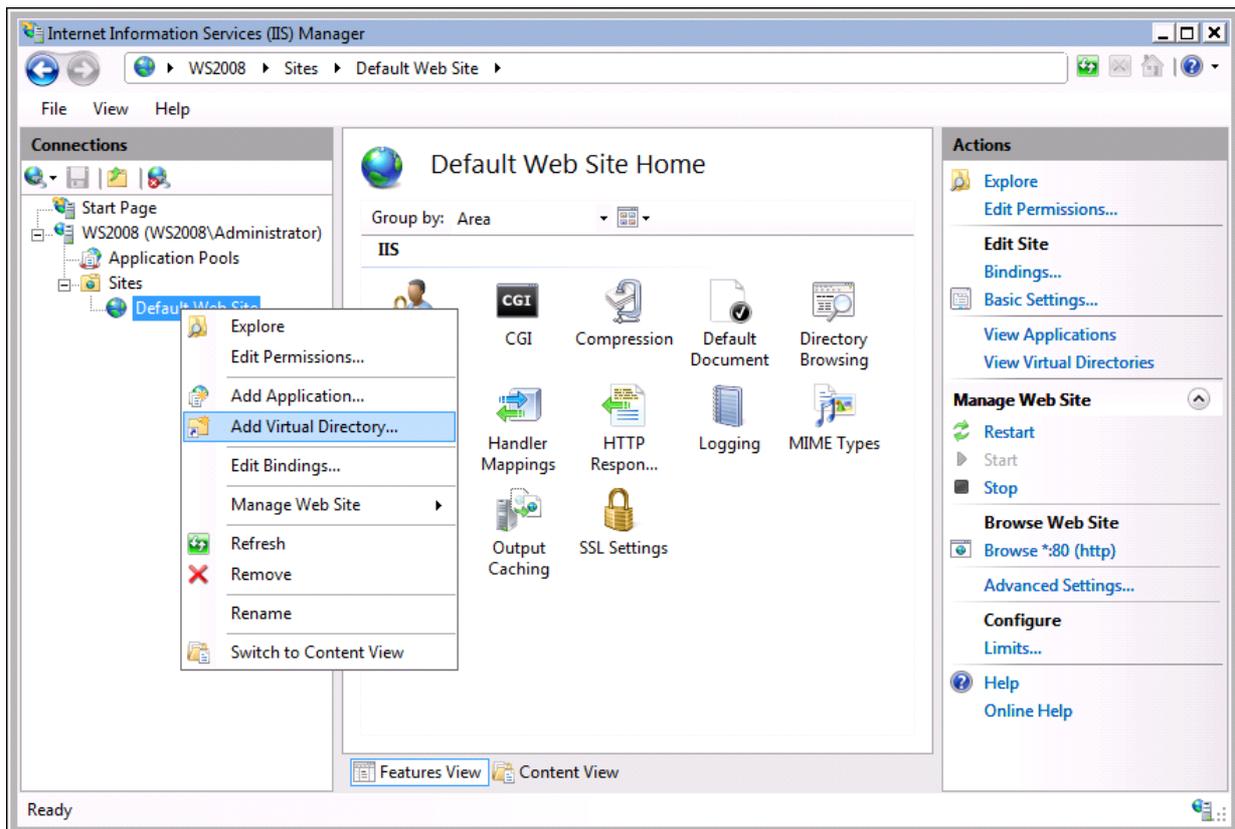


Double-click on the "ISAPI and CGI Restrictions" icon. In the overview that follows, click on the "Edit Feature Settings..." action item on the right side. This gives a dialog, Edit ISAPI and CGI Restriction Settings, and check the Allow unspecified CGI modules and/or Allow unspecified ISAPI modules.



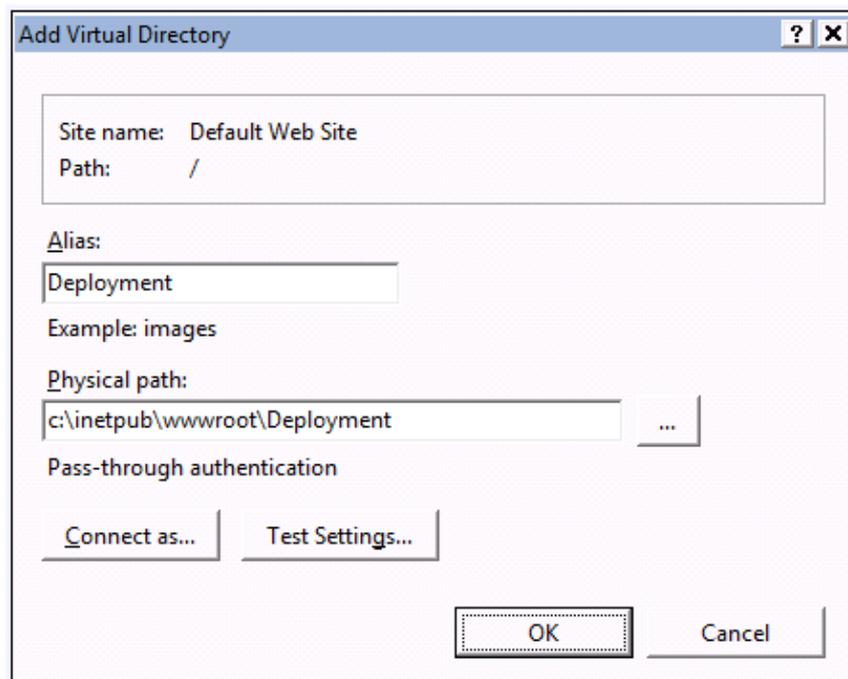
Once CGI and/or ISAPI Settings are possible, we can move down the tree on the left, and select the web site where we want to deploy the application.

Right-click (in this case on the Default Web Site) and select "Add Virtual Directory"

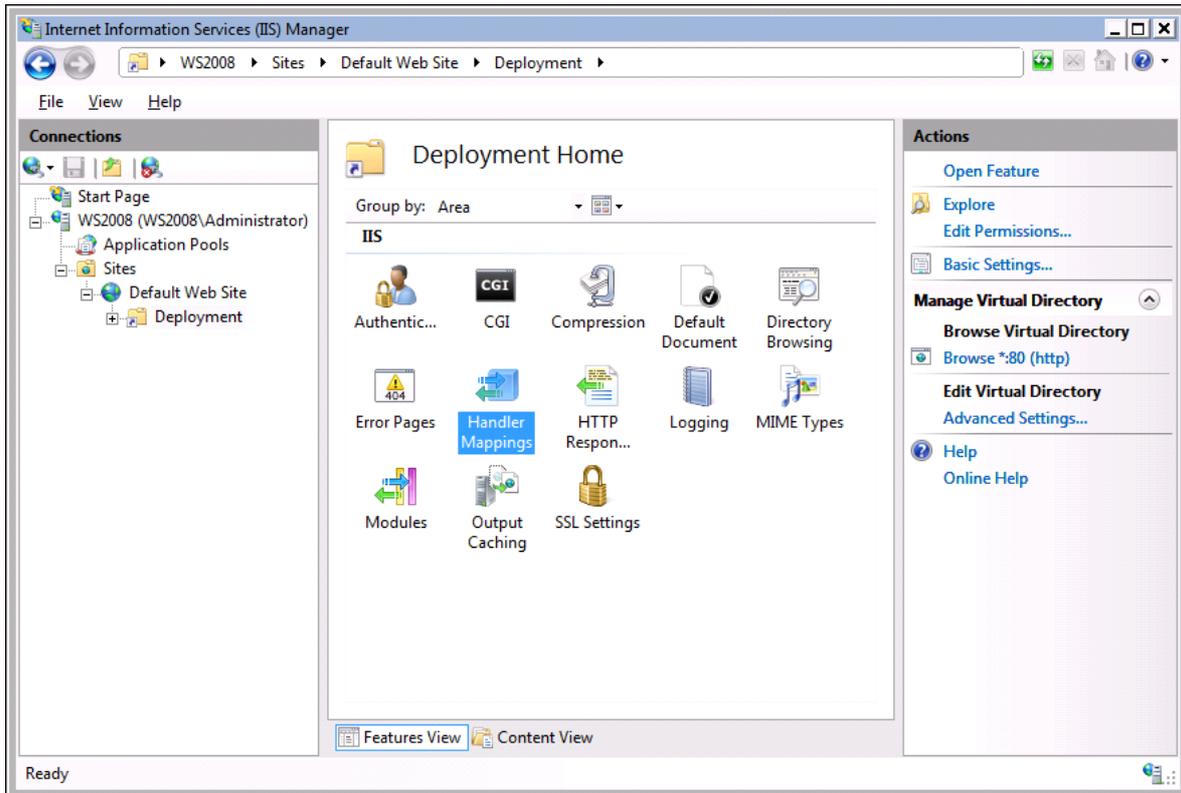


In the dialog that follows, specify the alias (Deployment in this case) as well as the physical path.

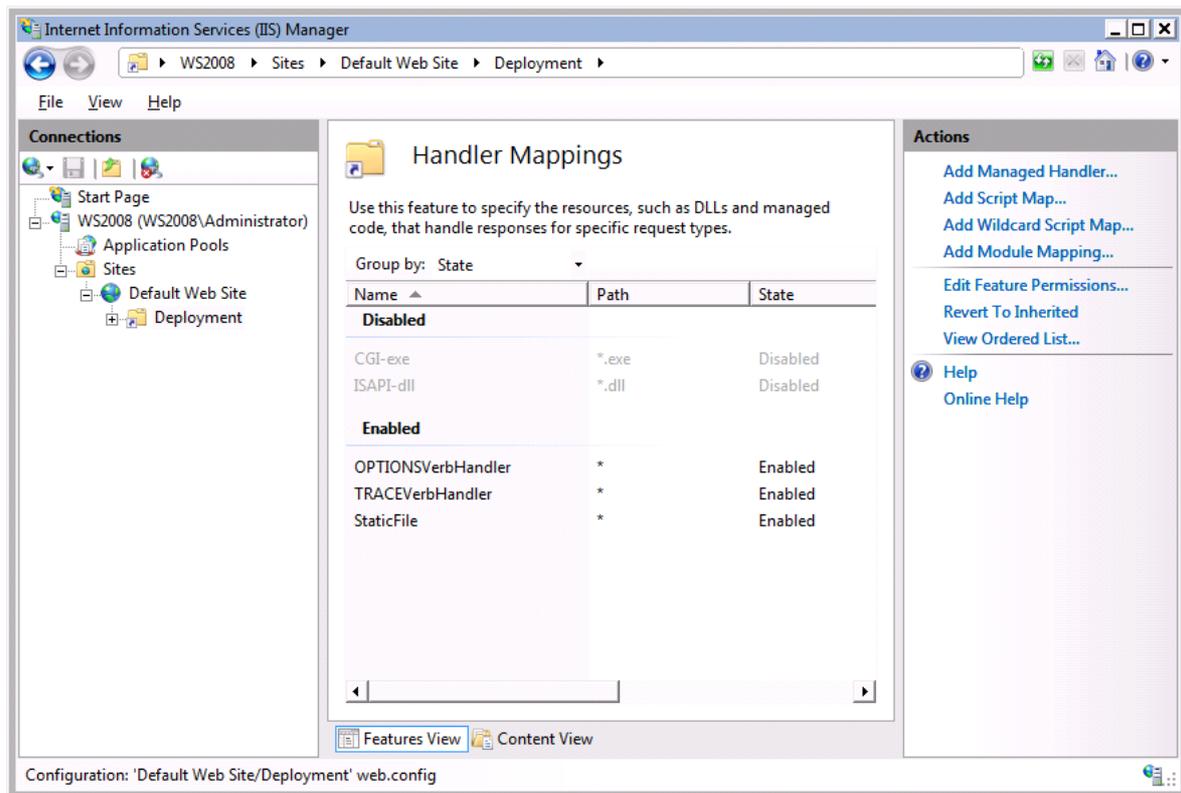
In our case, the physical path is c:\inetpub\wwwroot\Deployment. Make sure that the directory exists (otherwise you will get an error message).



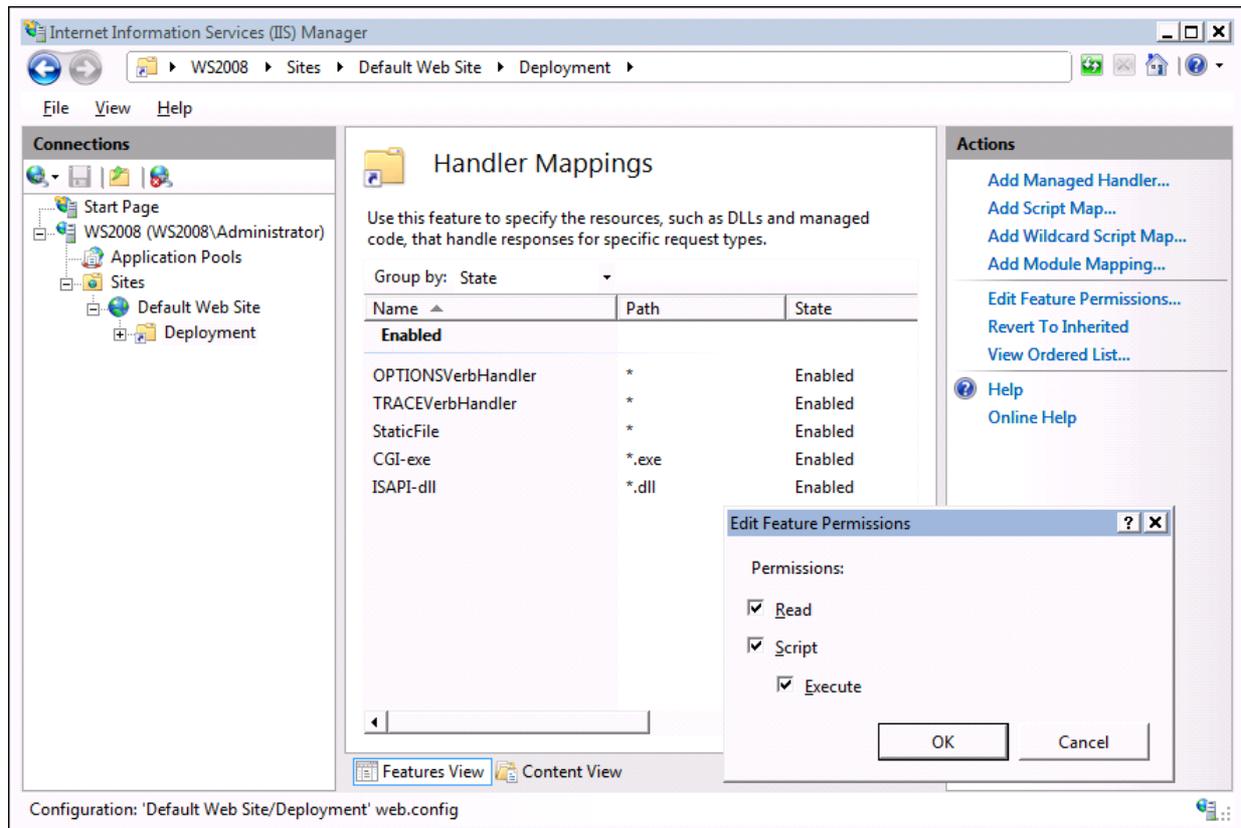
When the virtual directory is created, make sure to select it in the left side of the IIS Manager window, and then in the middle pane select the Handler Mappings.



Click on the "Edit Permissions..." action item, which changes the screen to the Handler Mappings contents, which shows the CGI.exe and ISAPI.dll features disabled by default.

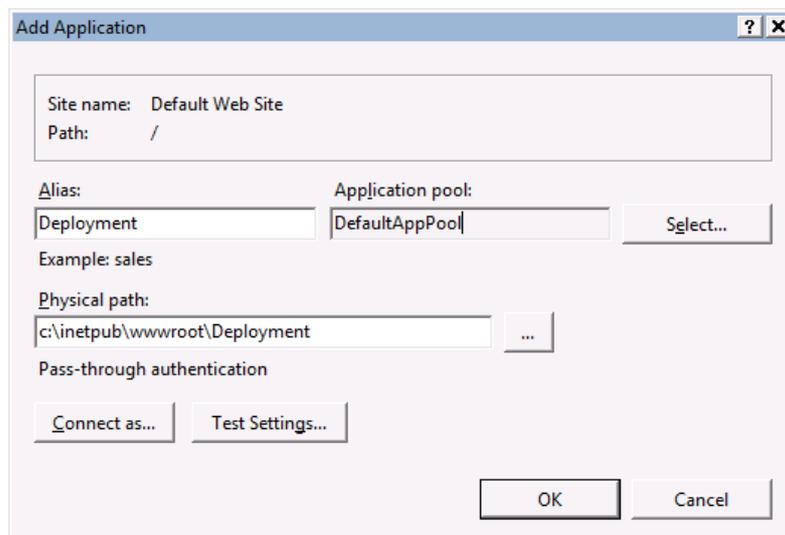


Click on the "Edit Feature Permissions..." action item, and in the Edit Feature Permissions dialog that follows check the Execute right item.



This completes the steps to enable CGI and/or ISAPI Extensions on IIS7. We can now deploy CGI and/or ISAPI extensions in the Deployment virtual directory on the default web site.

Note that apart from a virtual directory – which is enough for CGI or ISAPI Extensions – you can also create an application (using the "Add Application" context menu on the web site). This gives you a similar dialog, but with a few more options such as an Application pool that can be used to manage the CGI and ISAPI extensions in the application directory.



This is mainly helpful when working on ASP.NET projects by the way.

As result of your deployment, on either Windows Server 2003 or Windows Server 2008, you should now be able to run the WebBrokerCGI.exe (or ISAPI DLL) from the virtual directory on your domain (or IP-address, if you don't have a domain registered).

Note that apart from deploying the application, you may also need to deploy the database or change the connectionstring to the databases your WebBroker application uses.

IntraWeb Deployment

Apart from the IntraWeb application itself – whether it's a StandAlone, Service or ISAPI Extension application – you often need to deploy additional files.

Files, Templates and Cache

Specifically, if you've have files in the Files subdirectory, then you need to deploy this Files directory together with your IntraWeb executable.

Also, when you make use of templates, which are often placed in the Templates subdirectory of the IntraWeb application directory, then you obviously need to deploy the Templates directory together with your IntraWeb executable as well.

Apart from deploying the Files and Templates directories to the web server machine, you must also ensure that the IntraWeb Application has read access to the Files and Templates directories.

Finally, the IntraWeb application will automatically (try to) create and use a Cache subdirectory in the IntraWeb application directory. Inside this Cache subdirectory, the IntraWeb application must be able to create, read and delete files (i.e. Full access). If you are unsure about the IntraWeb application being able to create the required Cache subdirectory, then you can create it yourself (or ask your ISP to create it) and give the required rights for this subdirectory to the IntraWeb application.

Database Drivers

When you use database access in your IntraWeb application, you may need to deploy the required database drivers as well. I'm assuming that the database itself it already installed and deployed, but on the web server you may need at least the database client software in order to connect (locally or remotely) to the database. Note that it's always recommended to deploy the database on a different machine than your web server.

The reason is the fact that a web server is directly connected to the internet, and no matter how secure the web server is protected, there is always a chance that the web server machine is compromised. And if your database resides on that machine, then the intruders will have direct access to your database (which has happened to an e-commerce vendor who stored their clients' information including credit card numbers in a database on the web server – until the web server was hacked and the database with over 15,000 credit card numbers was copied. This particular vendor will never be honoured by a visit or purchase from me, that's for sure).

The recommendation is to place your database on a database server behind the firewall, connecting to the web server using a second network card, but not directly accessible from the internet. That way people who hack the web server may still be able to access the database (if they obtain the right credentials), but at least it's no longer possible to simple copy the database files.

DBX4 Drivers

Regardless of where you place the database itself, on the web server you may have to deploy the database client software, as well as the Delphi database drivers. Especially for dbExpress (DBX4) you will need to deploy additional files since the current dbExpress drivers that are included with Delphi 2007 and CodeGear RAD Studio 2007 in the box are still DBX3 drivers that cannot be linked-in with your IntraWeb executable. Also, when you use dbExpress and TClientDataSet components, you must either deploy the MIDAS.DLL with your IntraWeb application, or add the MidasLib unit to the uses clause of your application in order to link the functionality inside your application.

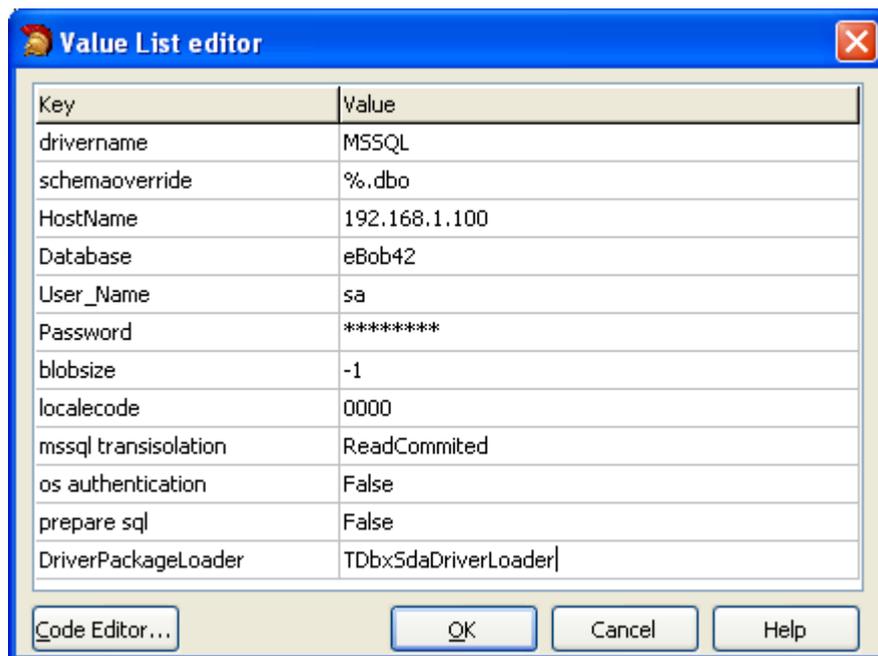
DBX Trace / Pool Connections

When we use the DBXTraceConnection or DBXPoolConnection we also have to deploy the dbxdrivers.ini and dbxconnections.ini file. The reason is that neither of these two delegate drivers are directly connected to a TSQLConnection component that holds their parameters, so the DBX4 framework has no way to know what the DBXTraceConnection consists of, for example. So far, I found no other way other than to include the dbxdrivers.ini and dbxconnections.ini files to the list of files to deploy. The good news is that you only need [DBXTrace] and [DBXTraceConnection] entries in the dbxdrivers.ini and dbxconnections.ini files.

Core Lab DBX4 Drivers

Since it's a bit of a shame that you cannot produce IntraWeb executables with the embedded DBX4 drivers from CodeGear, let's now examine Core Lab - one of the third-party vendors that produces database access drivers, including dbExpress drivers for Delphi and RAD Studio. They offer a DBX4 driver for MS SQL Server that I will use in the example now (you can download a 30-day trial edition from the www.crlab.com website to test it for yourself if you wish).

Once you've installed the dbExpress driver for SQL Server from Core Lab, you can change the LibraryName of any TSQLConnection component in your data module from dbxmss30.dll to dbexpsda40.dll (the DBX4 driver from Core Lab). Then, add a new attribute DriverPackageLoader to the Params of the TSQLConnection component, and give it the value TDbxSdaDriverLoader.



Finally, add the DbxSdaDriverLoader unit to the uses clause of your main IntraWeb project file. You may also want to add the MidasLib unit to the uses clause, so you also do not have to deploy the MIDAS.DLL file.

Before you can recompile the project, you must now also add the LITE;DBX40 conditional defines to the project, which can best be set in the Project Options dialog, where you also need to make sure the Core Lab source files can be found in the search path, by adding C:\Program Files\CoreLab\Dbx5da to the search path.

The result will be an executable with the embedded DBX4 driver loader (DbxSdaDriverLoader) and the dbexpsda40 DBX4 driver from Core Lab for SQL Server, that you can deploy as stand-alone ISAPI DLL.

Apart from DBX4 drivers for SQL Server, Core Lab also offers dbExpress drivers for Oracle, MySQL, and InterBase/Firebird, as well as native (non-dbExpress) drivers. Check out their website at <http://www.crlab.com> for more details.

Summary

In this section, I've explained what the different IntraWeb project targets consist of, and what the advantages and disadvantages are when it comes to deployment. I've also explained and demonstrated how to create virtual directories on Windows Server 2003 and 2008, and make them ready for IntraWeb deployment.

Finally, I've also covered additional files and directories required for deployment, and paid some special attention to database drivers and the CoreLab DBX4 drivers which can be linked-in with your IntraWeb executable.