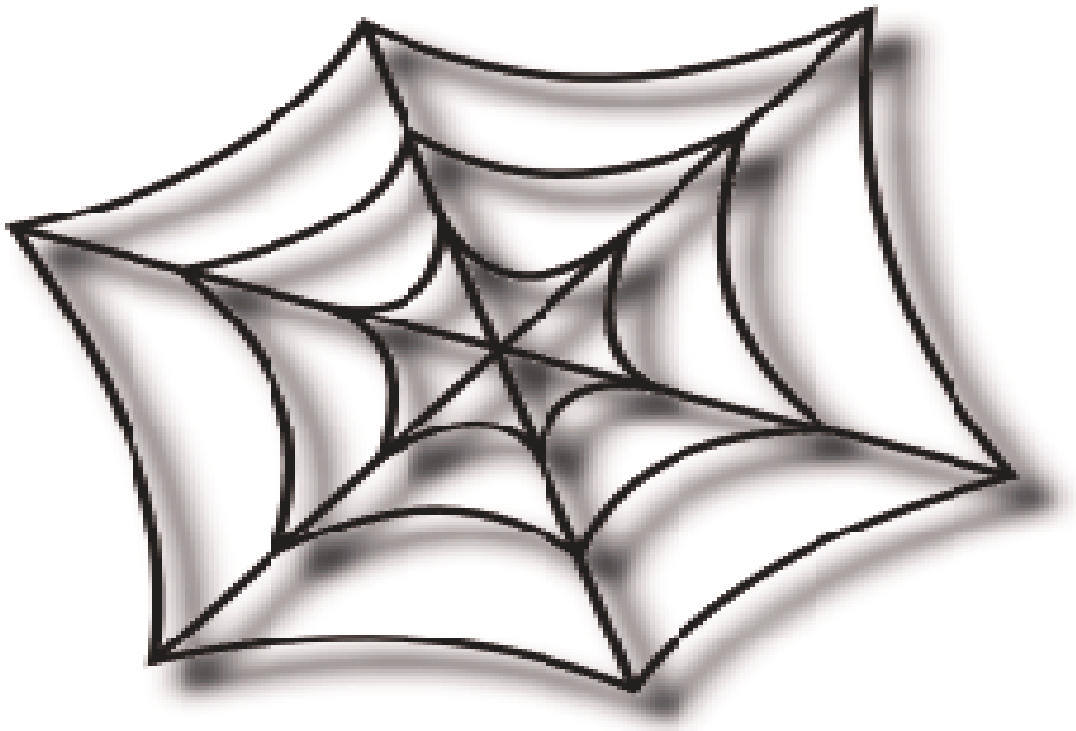
IntraWeb Manual



IntraWeb

A revolutionary approach to web application development

"Makes development of web applications as easy as falling off a log..."

PC Plus Magazine, June 2002

IntraWeb

Table of Contents

Foreword 1

Part I Welcome 3

1 Other Documentation 3
2 How IntraWeb Works 3
3 Limitations of the Evaluation Version 3
4 Technical Support 4
5 License 4
6 Credits, Acknowledgements and Copyrights 5
7 Requirements 6
Overview 6
Supported Browsers 7
Development Environments 8

Part II What's New 10

1 Overview 10
2 PDA Support 11

Part III Migration Guide 13

1 6.0 to 7.0 13
2 5.1 to 6.0 13
3 5.0 to 5.1 14

Part IV Installation 18

1 Installation of License Keys 18
2 Visual Studio.Net 18
3 Delphi / C++ Builder 18
4 Kylix 18
5 Upgrading to a new version 18
Delphi 7 Users 19

Part V Demos 21

1 Quick Start 21
2 Building A Demo Step By Step 21
3 Features 31
4 Guess 31
5 GuessWB 32
6 GuessMulti 32
7 FishFact 32

8 FishFactDM 32
9 Phoenitics Customer Profiler 32
10 StandAloneSSL 32
11 WebSnapSurvey 32
12 Custom StandAlone 32
13 Back Button 33
14 Page Forms 33
15 FishMarket 33
16 Die, Fly ! Die ! 33
17 WebMail32 34

Part VI Debugging 36

1 Getting Started 36
2 Debug Output 36
3 Detecting Errors on Startup 36
4 Command Line Parameters 36

Part VII Development 39

1 Rethinking the User Interface 39
2 Writing Your First IntraWeb Application 39
3 Images and Graphics 41
4 Extending IntraWeb 42
5 Working with COM 42
6 Working with ClientDataSet Components 43
7 Working with PDA 43
8 Miscellaneous 44

Part VIII Form Management 47

1 Working with Forms 47
2 Update Mode 47
3 IntraWeb Frame 48
4 Intraweb User Control 49
5 Visual Form Inheritance 49
6 Managing Forms 49
7 Form List 50
8 Showing Forms 50
9 Hiding Forms 51
10 Destroying Forms 51
11 Passing Data Between Forms 51
12 Note for C++ Builder users 57

Part IX State Management 59

1 Inherent State ... 59
2 Restrictions ... 59
3 Safe Storage ... 59
4 Complex State and the Back Button ... 62
Using the OnBackButton Event ... 64

Part X Session Management 66

1 WebApplication Object ... 66
2 Referencing the Session ... 66
3 Lifetime ... 66
4 Implementation ... 67
5 Storing Additional Data ... 67
6 Session Related Events ... 67
7 Memory Consumption ... 68
8 Component Reference ... 68
9 How does Session Management Work? ... 68
URL ... 68
Cookies ... 69
Hidden Fields ... 69

Part XI Layout Managers and Templates 71

1 What is a Layout Manager? ... 71
2 Form Layout ... 71
3 HTML Templates ... 71
4 System Templates ... 73

Part XII Server Controller 75

1 What is the Server Controller? ... 75
2 Properties ... 75

Part XIII Writing Custom Components 77

1 Overview ... 77
2 Under Construction ... 77
3 Source Code ... 77
4 Core Server ... 77
5 Third Party Program ... 77

Part XIV Javascript 79

1 Overview ... 79
2 Areas of Implementation ... 79

3 Using ScriptEvents 79
Writing JavaScript events 82
More Script Events 84
4 Javascript Functions 85
Common Functions 85

Part XV Page Mode 88

1 Introduction to Page Mode 88
2 Working with Page Mode 88
3 IntraWeb and Websnap 92
Creating the Demo 93
Running the Demo 99
4 IntraWeb Pages (IWP) 102
Introduction to IWP 102
Creating the Project 103
Preparing the server 103
Creating an IWP page 105
Creating an IWP form 106
Deploying the files 106

Part XVI IntraWeb Caching Mechanism 109

1 Caching Files 109

Part XVII Deployment 111

1 Installation 111
Overview 111
External Files 111
Permissions 111
ISAPI 112
Deploying in IIS 112
ISAPI Utilities 115
ISAPI Hosting 115
Common ISAPI Issues 116
Useful ISAPI links 117
2 Methods 117
Notes 117
Stand Alone 117
ISAPI / NSAPI / Apache DSO / CGI / Win-CGI 118
Windows Service 119
3 Launch URLs 119
Linking to IntraWeb Applications 119
Sessions 119
Passing Parameters 120
4 Converting Project Types 121
Converting from Standalone to ISAPI in Delphi 121
Converting from StandAlone to ISAPI in CBuilder 122
Converting from ISAPI to StandAlone in Delphi 123
5 Additional Linux Information 124
Overview 124

6 Application ports 124

Part XVIII Performance Evaluations 126

1 Tips 126

Part XIX Scaling IntraWeb Applications 128

1 Scaling Methods 128
2 Distributing the IntraWeb Application 128

Part XX Secure HTTP / SSL 131

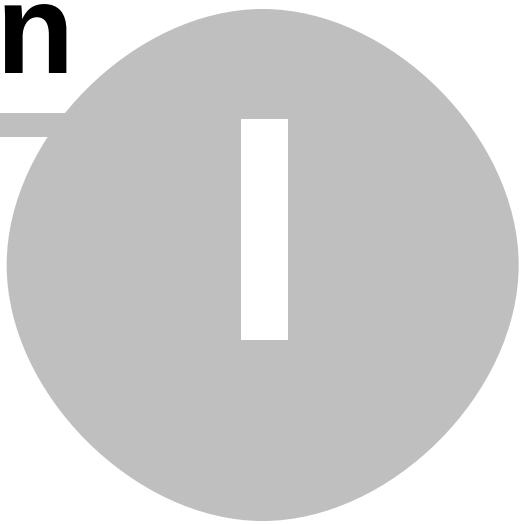
1 Introduction 131
2 Enabling SSL 131
3 Converting Certificates to PEM Format 131
4 Example 132

Index 133

"In a nutshell, IntraWeb does things the Delphi way. ..... So am I impressed? Yes, very much so. ..... As I've examined WebSnap and compared it with IntraWeb, it's clear to me that IntraWeb is the sort of technology that I would have expected to see from Borland but didn't."

Delphi Magazine, March 2002

Section



1 Welcome

IntaWeb is a revolutionary new way to create web-based applications. Built upon Web Solution Builder and earlier versions of IntaWeb, it extends the technology of both of these, providing an excellent tool for creating Internet, Intranet and Extranet applications in a quick and easy to maintain manner.

Many web-based development tools require the user to have knowledge of CGI scripting, state tracking and complex client configurations. IntaWeb does away with all these hassles and overheads. By simply creating the application using the component suite within Delphi or Visual Studio, and later registering it on the server, clients can access the application using any browser that complies with HTML version 4. These include the latest versions of Netscape and Internet Explorer. Both of these have been fully tested with IntaWeb and are 100% compatible. No HTML, CGI or JavaScript coding is required; all the coding is done using Delphi or C# and VB.NET for Visual Studio. For further flexibility, the application can also be run as a stand-alone executable like any other Desktop application, providing debugging capabilities.

1.1 Other Documentation

Be sure to check the IntaWeb FAQ (available on the Atozed Software website) as well as the information available on the website itself. A lot of documentation is contained there that is not in the manual or the help file and to keep it accurate and current we have not duplicated it.

This document is designed to be a manual, not a reference guide. The IntaWeb Component Reference help file should be consulted when a reference for properties, events, methods and components is needed.

1.2 How IntraWeb Works

IntaWeb works much like a normal executable application, with the exception that the user interface is a web browser instead of a window. After placing the application on a web server, a user can run an instance of the application by using a URL to start a session. The user's information will be tracked by the instance of the application in use, thus preventing loss of the information or mixing it up with other user information. For each user, new session information is created and tracked automatically and transparently to the developer. The overhead is low and the capacity of an IntaWeb application is similar to that of other web solutions such as ISAPI, CGI, or ASP.

IntaWeb is designed to build any sort of web-based application, whether it is a simple data entry form, a poll, or a complex application where clients have to be "logged in" for an extended period of time.

1.3 Limitations of the Evaluation Version

Limitations of the evaluation edition have been designed in such a way so that you can fully evaluate your application with no time limits and no development restrictions. The only restrictions in the evaluation edition exist to hinder deployment.

Unique Port Numbers

When using the evaluation version of IntaWeb, the port numbers that your IntaWeb application listens will be unique upon each execution. Any port settings you may specify will be ignored. Testing is facilitated in evaluation mode by use of the execute menu item as it automatically adjusts its URL for the changing port.

IP Restriction

In the evaluation version, IntaWeb applications only listen on the IP 127.0.0.1. No requests from other IP addresses will be answered.

No Services
IntaWeb applications cannot be installed or run as services in the evaluation version. Attempts to do so will result in errors.

No SSL
SSL is disabled in the evaluation version.

No Deployment License
You may not deploy any applications created with the evaluation version.

1.4 Technical Support

Up-to date support information is available at http://www.atozed.com/.

1.5 License

Single User License agreement
This is a legal agreement between you the end user, and Atozed Software (hereafter referred to simply as Atozed). By using this software package you are agreeing to be bound by the terms of this agreement. If you do not agree with the terms of this agreement, promptly uninstall this software.

Grant of License - Atozed grants to you the right to use one copy of the enclosed software for a single developers use. This includes the ability to install on multiple computers so long as the installation is only used by the same single developer. In plain English this means you can install this software on your desktop and your laptop so long as you are the only one who uses the software. You may not install on multiple computers for multiple developers to use. Furthermore you may not install on one computer for multiple developers to use aside from normal debugging purposes and peer review.
You may make one copy of the software for backup purposes for use on your own computer. The original software must be backed up in unmodified form.
You may not network the software or use it on more than a single computer or computer terminal at any time, unless a copy is purchased for each developer on the network that will use the software. You may not rent or lease the software, but you may transfer the software and accompanying written material and this license to another person on a permanent basis provided you retain no copies and the other person agrees to accept the terms and conditions of this agreement.
THIS SOFTWARE MAY NOT BE DISTRIBUTED, IN MODIFIED OR UNMODIFIED FORM, AS PART OF ANY APPLICATION PROGRAM OR OTHER SOFTWARE THAT IS A LIBRARY-TYPE PRODUCT, DEVELOPMENT TOOL OR OPERATING SYSTEM, OR THAT MAY BE COMPETITIVE WITH OR USED IN LIEU OF THE PROGRAM PRODUCT, WITHOUT THE EXPRESS WRITTEN PERMISSION OF Atozed.
This license does include the right to distribute applications using the enclosed software provided the above requirements are met.

Term - This agreement is effective until you terminate it by destroying the software, all copies and backups. It will also terminate if you fail to follow this agreement. You agree upon termination to destroy the software, together with all copies thereof including backups.

Copyright - The software is owned by Atozed and is protected by International laws and treaties. Therefore, you must treat the software like any other copyrighted material.

Warranty Clause

Limited Warranty - Atozed warrants that the software will perform substantially in accordance with the accompanying written materials. Atozed does not warrant that the functions contained

in the software will meet your requirements, or any operation of the software will be uninterrupted or free of errors.

Disclaimer of Warranties - Atozed disclaims all other warranties, either expressed or implied, including but not limited to implied warranties of merchantability of fitness from particular purpose, with respect to the software and accompanying written materials. Atozed will have no consequential damages. In no event, shall Atozed or its suppliers be liable for damages whatsoever, (including without limitation, damages for loss of business profits, business interruption, loss of business information, or any pecuniary loss), arising out of the use or the inability to this product, even if Atozed has been advised of the possibility of such damages.

Sole Remedy - Atozed holds no liability for remedy.

U.S. Government Restricted Rights - This software and documentation are provided with restrictive rights. Use, duplication or disclosure by the Government is subject to restrictions set forth in Section c(1)(ii) of the Rights and Technical Data in Computer software clause at 52.227-7013.

1.6 Credits, Acknowledgements and Copyrights

Every attempt has been made to provide credit to the proper sources. However in many cases multiple sources have been used only for consultation. In other cases original authors are not known, or no licensing information was provided.

IntaWeb
IntaWeb is a commercial product and fully protected by International copyright laws and treaties.

Help and Manual
Help and Manual is used to produce this and other help files and documentation.
http://www.ec-software.com/

Inno Setup
The IntaWeb installation is performed using Inno Setup which is Copyright © 1998-2001 Jordan Russell with portions by Martijn Laan.

The A-Team and TeamZed
Thanks to the A-Team and TeamZed for their continued support of IntaWeb.

Indy
IntaWeb uses Indy to provide much of its functionality. Indy requires that its copyright be included. Please note that this copyright only applies to the Indy code itself. More information on the licensing of Indy can be found at http://www.nevrona.com/indy/license.html.

Copyright
Portions of this software are Copyright (c) 1993 - 2001, Chad Z. Hower (Kudzu) and the Indy Pit Crew - http://www.nevrona.com/Indy/

License
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the

following disclaimer in the documentation, about box and/or other materials provided with the distribution.

No personal names or organizations names associated with the Indy project may be used to endorse or promote products derived from this software without specific prior written permission of the specific individual or organization.
THIS SOFTWARE IS PROVIDED BY Chad Z. Hower (Kudzu) and the Indy Pit Crew "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Original TIWMenu author
Thanks to the person who originally authored this component. However they forgot to mark their name in the .pas file. :)

Jason Southwell
For contribution of several components and starter components.
Contribution of TIWDataModulePool in IntaWeb 6

Thomas Brattli
http://www.dhtmlcentral.com
For eXperience DHTML cool Menus which are the basis for TIWMainMenu.

Motty Adler
For contribution of the idea and the source that TIWTimer is based on.

L. Relihan & P. Kerl
Authors of the PacText Java applet used in the features demo.

Paul Stockton and Anthony Lowery
For their contribution to the calendar code.

Others
Thanks to all the other parties who have assisted with both small and large contributions that we may have overlooked.

Jorrit B. Jongma
Contribution of compression routines and classes for HTTP compression in IntaWeb 6

1.7 Requirements

1.7.1 Overview

The only requirement is that users of IntaWeb developed applications have browsers that are HTML 4 compatible *, since extensive use of HTML 4 and JavaScript are made. IntaWeb has been extensively tested with Netscape and Internet Explorer and is supported with Mozilla, Netscape 6 and higher and Internet Explorer 4.0 and higher.

NOTE: If you want Netscape 4 support you should use IntaWeb version 4. Which we will continue to

maintain and support. As of version 5.1, IntaWeb supports HTML 3.2 in PDA version. Therefore, Netscape 4 is supported to a certain extent (i.e., no use of CSS or JavaScript).

HTML 4
IntaWeb uses HTML 4 and style sheets to achieve the coordinate placement of items and other features. Usage of templates or page mode can eliminate the need for style sheets.

JavaScript
JavaScript is used to allow many advanced client features. JavaScript also allows IntaWeb to control the browser and rendered pages. JavaScript is only required for Application Mode.

PDA (HTML 3.2) support
Intraweb 5.1 introduces PDA support. Based on HTML 3.2, 3.2 mode allows you to develop applications that are compatible with most PDA devices. 3.2 mode does NOT require JavaScript or CSS, making it entirely compatible with any device that supports HTML 3.2. Please note that to use 3.2 you are restricted to the 3.2 controls that appear on the palette. Most of these controls are the same as the standard IW controls, however due to HTML 3.2 restrictions and the lack of JavaScript, they offer in some cases less functionality than their standard counter-part.

1.7.2 Supported Browsers

Even with HTML and JavaScript standards in place, the browsers differ in many areas. IntaWeb adjusts for these differences automatically. IntaWeb generates the appropriate HTML and JavaScript code for the browser. IntaWeb even knows about certain bugs in specific versions of each browser and works around them dynamically. In other cases, output for each browser is optimized. See the section on Browser Implementations for more information.

Browser Implementations
Even with HTML and JavaScript standards in place, the browsers differ in many areas. Much of this is because browsers often make extensions before such features are adopted as standards. Often different browsers implement similar features in incompatible ways.

IntaWeb adjusts for these differences automatically. IntaWeb generates the appropriate HTML and JavaScript code for the browser. IntaWeb even knows about certain bugs in specific versions of each browser and works around them dynamically. In other cases, output for each browser is optimized. IntaWeb performs all of this transparently to you and without using Java, ActiveX, or any plug-ins.

Internet Explorer
Internet Explorer versions 5, and 6 are supported.
Internet Explorer 4 is not officially supported anymore. However, any HTML 3.2 application should run without any problems on Internet Explorer 4.

Netscape
Netscape 7 is supported.
Netscape 6 is not supported. The version 6 of Netscape has serious problems with HTML rendering and JavaScript implementation. A better version is available for download on the Netscape site.

Although Netscape 6 is not officially supported, most of the applications created with IntaWeb will run correctly on it. There are known problems with Netscape 6 when using partial updates, when using regions, when caching files and when trying to use the TIWMenu control. There might be, however, more issues. When developing for Netscape 6, please make sure to test your application with a newer version of Netscape (7 or greater) before searching for bugs.

Mozilla
Mozilla is supported. Please make sure you're running the latest Mozilla update.

Opera
Opera 7 is fully supported.
Due to the amount of restrictions Opera 6 has with support for JavaScript, certain characteristics are not supported currently in Opera. Many of these have to do with Anchors and alignment. Left and Top positioning is supported with anchors, however, currently Right and Bottom are not. Other functionality is supported in Opera.

Other Browsers
Any browser that supports HTML 3.2 rendering is supported. However, HTML 3.2 is limited in many aspects, most noticeable being the lack of Javascript support.

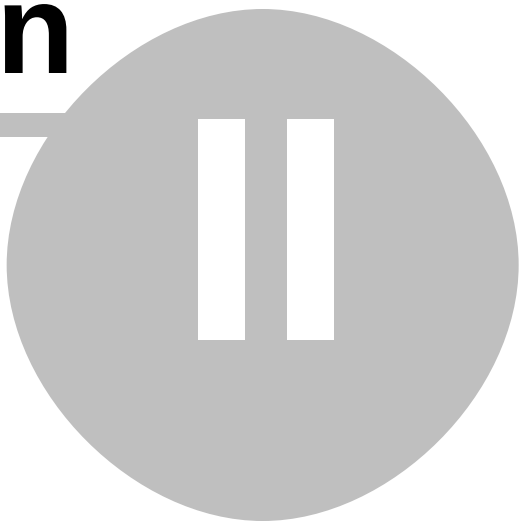
1.7.3 Development Environments

When purchasing IntaWeb Enterprise Edition (does not apply to special offers), support is provided for the following development environments:

- Delphi 5 (Pro and Enterprise)
- Delphi 6 (Pro and Enterprise)
- Delphi 7 (Professional, Enterprise and Architect)
- C++ Builder 5
- C++ Builder 6
- Kylix 2
- Kylix 3
- Visual Studio .NET 2003

There is also a Java edition that supports JBuilder, which is sold as a separate product

Section



2 What's New

2.1 Overview

IntaWeb 7 introduces many new features, including:

Hints Improved
In IW 7 it is no longer necessary to add an ESCAPE character when using the single quote(') character in the hint's text.

Raw Text
The TIWURL, TIWURL32, TIWLink, TIWHyperLink32 controls now have the RawText property. This property refers to how the caption of those controls is rendered as HMTL. For more information on the RawText property please see the reference guide.

TIWTreeView Control Improved
The Font property of the TIWTreeView control contains now a CSSStyle member, that can be edited in the designer.

Unknown Browser Detection
In IntaWeb 7, unknown browser detection functions at form level as well as at server level. This means that, when an unknown or unsupported browser is detected at form level, the application will redirect at the URL specified in the server controler's property UnknownBrowser.

Individual Session Timeout
In IntaWeb 7, when a session is started, it's timeout is defaulted to that of the global one, defined in the Server Controller. (For more information about the session timeout in the Server Controller, refer to the TIWServerControllerBase.SessionTimeout in the Reference Guide.) Later on, the session timeout can be set individually.

DeviceNotSupported Property in Server Controller
When an unsupported device is accessed (such as a PDA with an WML device), the application redirects to the location specified by this property, and if the property is left blank, an exception with the message "Device not supported" is raised.

Small Changes:

- The ConvertSpaces property of TIWCustomText and TIWCustomText32 is defaulted to false. The ConvertSpaces property indicates wether spaces contained in the Text of the control should be rendered as ' ' or ignored.

Features introduced in 6:

- Partial/Progressive page update
- Scrollable regions
- Slow connection simulation
- Server side resizing
- IWP file processing
- Templates/subtemplates with TIWRegion and TFrame
- SSL on demand
- .NET Common Code Base
- HTTP compression
- Improvements to align/anchors
- Font size measured in pixels, instead of points

- RGB entry in Color property editor

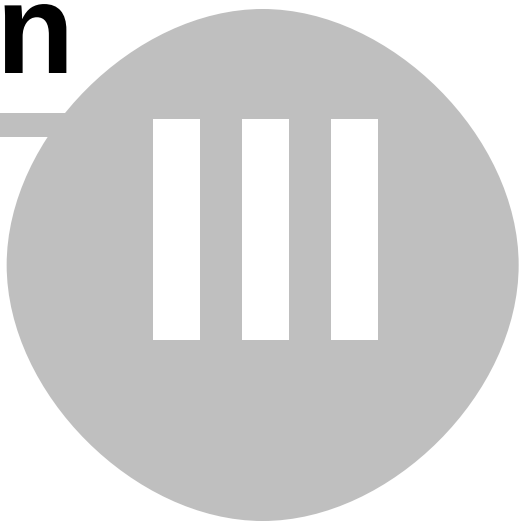
2.2 PDA Support

Intraweb 5.1 introduces support for hand-held devices (referred to from now on as PDA). By using HTML 3.2 and no additional JavaScript or CSS, IntaWeb allows you to develop robust and efficient applications for any HTML 3.2 compliant PDA device.

The following HTML 3.2 components are now available:

- TIWLabel32 - label control
- TIWEdit32 - edit box
- TIWButton32 - push button
- TIWList32 - HTML list
- TIWHRule32 - HTML horizontal rule
- TIWListBox32 - listbox control
- TIWComboBox32 - combobox control
- TIWRadioGroup32 - radio group control
- TIWImage32 - image control
- TIWImageFile32 - image file control
- TIWURL32 - link control
- TIWGrid32 - grid control
- TIWMemo32 - memo control (HTML textarea)
- TIWText32 - text control
- TIWRectangle32 - rectangle control
- TIWCheckBox32 - check box control
- TIWHyperLink32 - link control
- TIWDBEdit32 - data aware edit box
- TIWDBCheckBox32 - data aware checkbox
- TIWDBComboBox32 - data aware combobox
- TIWDBLabel32 - data aware label control
- TIWDBListBox32 - data aware listbox control
- TIWDBLookupComboBox32 - data aware combobox used for field lookups
- TIWDBLookupListBox32 - data aware listbox used for field lookups
- TIWDBMemo32 - data aware memo control (HTML textarea)
- TIWDBText32 - data aware text control
- TIWDBImage32 - data aware image control
- TIWDBRadioGroup32 - data aware radio group control
- TIWTemplateProcessorHTML32 - template processor for HTML templates
- TIWPageProducer32 - page producer used to work with WebSnap in page mode
- TIWLayoutMgrHTML32 - layout manager

Section



3 Migration Guide

IntraWeb migration guides are maintained for historical purposes. If you are upgrading from older versions, you need to read each set of notes for each version.

Always back up your projects before attemting any conversions. Also always allow adequate time for conversions.

3.1 6.0 to 7.0

Upgrading to IntraWeb 7.0 from 6.0 takes less than 5 minutes for most projects. Simply load the Server Controller at design time, and click "Ignore". Now save and recompile. Unless you were using the TimeOutURL property, no other covnersion is required.

The TimeOutURL property changes to SessionTimeOutURL
The TimeOutURL property of the server controller changes its name to SessionTimeOutURL and is no longer of type string, but TIWFileReference.

Visual User Session
IntaWeb 7 introduces visual user session object. The user session object is created by the IntaWeb Project Wizard and it is in fact a Data Module. Some of the advantages of this approach are:

- components needed by the user session can be added directly by dropping them on the form.
- it can be used as a data module, and all the session management is done by the application
- new methods and members can be added to the user session, as before.

Previous user session objects are still compatible with IntaWeb 7, so there is no need of updating the user session to the new type unless taking advantage of the new features is wanted.

Note: Although the new user session can function as a data module, when using data module pooling, a data module is still created.

3.2 5.1 to 6.0

Changes required when upgrading from 5.1 to 6.0:

The BGColor property
For easier and intuitive usage, the BackgroundColor properties of the form (TIWForm) and of the grid cells of dynamic grid (TIWDynGrid) have been renamed to BGColor. The BGColor name has been chosen as it's used for background colors throughout the components of IntaWeb.

When loading forms the contents of the old BackgroundColor property will be lost, and you'll have to set the desired color again in the BGColor property. Same thing applies when loading forms that contain TIWDynGrid controls.

ISAPI Projects
This change was introduced in the late stages of 5.1. For new applications, the wizard will automatically create the correct DPR automatically. However, for existing projects, you need to add the following code to your DPR file (highlighted with bold):

```
    ISAPIApp,
    IWInitISAPI,
    ServerController in 'ServerController.pas' {IWServerController: TIWServerController},
    Unit1 in 'Unit1.pas' {IWForm1: TIWFormModuleBase};

{$R *.RES}
```

```
exports
  GetExtensionVersion,
  HttpExtensionProc,
  TerminateExtension;
```

Frames
You need to convert existing TFrame forms to IntaWeb frames. To do so, create a new IntaWeb Frame using the project wizard and then cut and paste your components on the new frame. More information on how to create a IntaWeb Frame can be found in the IntaWeb Frame chapter of this manual.

3.3 5.0 to 5.1

Changes required when upgrading from 5.0 to 5.1:

Whenever possible interfaces have remained the same. However is some situations it was deemed better to change interfaces for future expansion.

For the most part migrating from a 5.0 application to 5.1 is straightforward and this section is designed to assist you with this migration.

URLBase
To use URLBase, you need to access it via WebApplication.AppURLBase

TIWImage
The UseBorder property has been deprecated. The new BorderOptions property should be used instead.

RWebApplication
RWebApplication is now simply WebApplication. Scope determines whether this threadvar version or the property version is accessed.

Project Files
The project files have a new format and standalone debug executables are now separate projects from the standalone service executables. Examples of each can be seen by looking at the Guess demo.

A new Standalone project file should look like this:

```
program Guess;

uses
  Forms,
  IWMain,
  Main in 'Main.pas' {formMain: TIWFormModuleBase},
  ServerController in 'ServerController.pas' {IWServerController: TDataModule};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TFormIWMain, formIWMain);
  Application.Run;
end.
```

When converting Service applications, you need to add *IWInitService* to the *uses* clause of the DPR and replace the existing code with:

```
program GuessService;
```

```
uses
  IWInitService,
  Main in 'Main.pas' {formMain: TIWFormModuleBase},
  ServerController in 'ServerController.pas' {IWServerController: TDataModule};

{$R *.res}

begin
  IWRun;
end.
```

ISAPI and DSO projects have the following layout (DPR file):

ISAPI

```
library GuessDLL;

uses
  IWInitISAPI,
  Main in 'Main.pas' {formMain: TIWFormModuleBase},
  ServerController in 'ServerController.pas' {IWServerController: TDataModule};

{$R *.RES}

begin
  IWRun;
end.
```

DSO

```
library GuessDSO;

uses
  ApacheApp,
  IWInitApache,
  ServerController in 'ServerController.pas' {IWServerController:
                                              TIWServerControllerBase},
  Main in 'Main.pas' {formMain: TIWFormModuleBase};

{$R *.res}

exports
  apache_module name 'GuessDSO_module';

begin
  IWRun;
end.
```

When compiling for Apache 2, you need to make sure that you include IWInitApacheTwo in the uses clause as opposed to IWInitApache. If you create an Apache 2 application using the wizard, this is automatic.

NOTE: *You need to make the appropriate changes to the Delphi 7 VCL before your modules will work with Apache 2. This information can be obtained on Dr. Bob's website at* http://www.drbob42.com/Delphi7/Apache2040.htm

Setting the main form and server controller

For all types of projects (Standalone, ISAPI, DSO, Service), two **initialization** sections have to be added. The first one should be in the Main form of the project and indicates which form is the main form. For example, for the Guess project this would be:

```
initialization
  TformMain.SetAsMainForm;
```

SetAsMainForm is a class procedure of a TIWForm. When working on projects that support both HTML 4.0 and PDA (HTML 3.2), it is VERY IMPORTANT that both the main form for the 4.0 version and the 3.2 version each have an initialization section indicating that it is the main form (see

GuessMulti for examples).

The other initialization section is in regard to the server controller. This has to be set in the ServerController itself and it is present for new projects:

```
initialization
  TIWServerController.SetServerControllerClass;
```

Here **SetServerControllerClass** is again a class procedure of IWServerControllerBase.

OnNewSession
The OnNewSession event handler has a change in the signature. The new signature is:

```
    OnNewSession(ASession: TIWApplication; var VMainForm: TIWBaseForm);
```

Be sure to include *IWBaseForm* and *IWBaseControl* in the *uses* clause of the *ServerController* if converting from a previous version.

Session and DataModule Owners
In previous versions (prior to 5.1), the session owner would be *ASession* and then DataModule owner (if present) would be *AOwner*. In 5.1, these have changed to *nil* for the first and *Self* for the latter.

Deprecated properties on forms
It is recommendable that when you open a previously made project in versions prior to 5.1, you go through all the
forms and ignore any non-existent properties, before compiling your application.

TemplateProcessor property
This property has been renamed to *LayoutMgr*. Assign any TemplateProcessor components on your form to this
new property

Application Path
In previous versions of IW, you needed to use gsAppPath to obtain the application path. From 5.1 you can use WebApplication.ApplicationPath to obtain this.

TIWFile
To obtain the size of a file, you no longer can access the Size property of the file stream. Instead you will have to get it using the following method:

```
  THTTPFiles(WebApplication.FileList).GetSize(IWFile1.Name)
```

Where IWFile1 is the TIWFile control used to transfer the file.

Section

4 Installation

4.1 Installation of License Keys

The download for registered users with license keys is the same as the evaluation edition. If you have the evaluation edition already installed you merely need to enter your license key using the registration utility. The registration utility can be run from its icon in the IntaWeb program group.

4.2 Visual Studio.Net

Installation program automatically integrates IntaWeb into Visual studio environment. The following tabs are added to the toolbox: "Intraweb Controls", "Intraweb DB Controls", "Intraweb Components", "My IW User Controls". "Intraweb Controls" tab contains the standard IntaWeb controls, the "Intraweb DB Controls" tab contains the data-aware Intraweb controls, the "Intraweb Components" tab contains the layout managers components and the "My IW User Controls" contains the user defined controls. The Intraweb applications must be created using the Intraweb application wizard accessible from the New Projects window in Visual Studio .NET.

For both environments the installer also creates program groups, which can be accessed via the Start menu. The documentation is placed in this program group.

4.3 Delphi / C++ Builder

The installation will automatically integrate IntaWeb into Delphi. Six new tabs will be created on the component palette containing the IntaWeb components. One of them contains the non-database components (IW Standard), another contains the data-aware ones (IW Data), the third contains control components (IW Control), the fourth tab is for client side controls (IW Client Side), the fifth tab is for HTML 3.2 non-database components (IW Standard 3.2) and the sixth is for HTML 3.2 data-aware components (IW Data 3.2). A new tab will also be created in the Delphi repository. All IntaWeb applications should be created using the templates contained in the repository under the tab IntaWeb.

The installation process copies the appropriate files to the Windows\System directory and to sub folders of all the Delphi environments selected.

4.4 Kylix

Please see the Linux installation instructions which are linked from the IntaWeb download page.

4.5 Upgrading to a new version

Before upgrading to a new version:

For Delphi:

1. Uninstall previous versions of IntaWeb
2. Make sure you do not have left-over files on your system. Check for:
   - Intra*.bpl
   - Intra*.dcp
   - IW*.*
3. Remove any of the files found in step 2. Make sure you check your entire system.
4. Install the new version of IntaWeb

For Visual Studio .NET:

1. Uninstall previous version of Intraweb for .NET.

4.5.1 Delphi 7 Users

If you are using the version of IntaWeb that is included in Delphi 7, and you are entitled to an upgrade (from the version included to version 5.1), you need to perform the following steps to install IntaWeb:

1. Uninstall the version included in Delphi 7
2. Generate a new key request by using the Key Request Application that can be found on the Atozed website (www.atozedsoftware.com).
3. Install the new version and enter the key obtained from step 2.

Section V

5 Demos

5.1 Quick Start

Until you are ready to deploy, you should use the stand alone versions for development and testing. For evaluation purposes you should start with the simplest demo, "Guess". Guess is a very simplistic demo (Our version of "Hello World"), but will introduce you to IntaWeb with a simple application. After that you can then look at the other demos to see more advanced features of IntaWeb.

Many of the demos of Intraweb for Delphi have multiple project files such as Features / FeaturesDLL / FeaturesDSO. Features is the standalone version, FeaturesDLL is the ISAPI version, and FeaturesDSO is the Apache version. They are different project files but share the same units and aside from the project file the source code is identical. Finally you can finish up with the "Features" demo which is not a "functional" application per se but a demo that demonstrates many of the features of IntaWeb merely to demonstrate them.

There are also additional user contributed source and demos available from the IntaWeb download page on the Atozed website.

5.2 Building A Demo Step By Step

This chapter will follow the development of a simple demo for Intraweb step by step. The demo is called FormData and shows how IntaWeb forms can interact with each other during a session.

Description

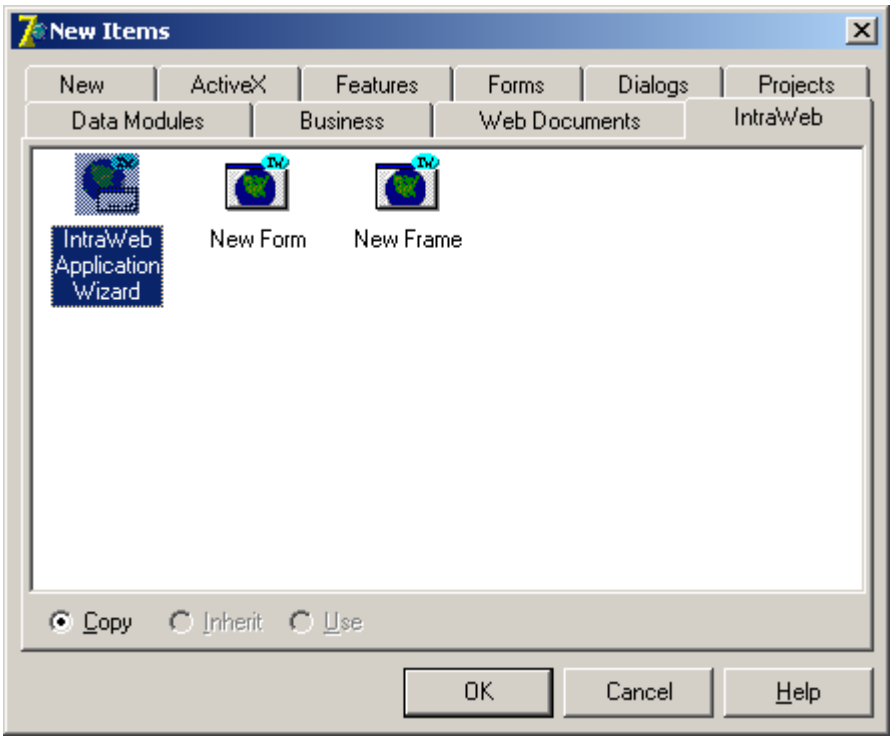
FormsDemo is a simple application which has two screens: first one displays an edit and a button. Clicking on the button will open the second screen which shows all data entered in the edit field of the first form and the number of times the user has visited the second screen. This is an stand alone application developed with IntaWeb.

How Does The StandAlone Application Work?

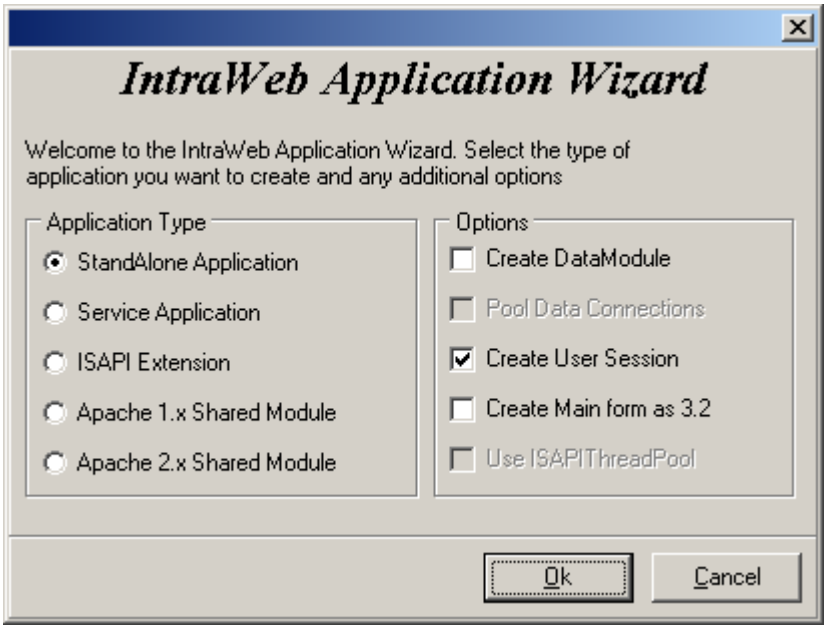
A standalone application is an executable that listens on port defined by the user and serves back HTML pages.
IntaWeb can build this kind of applications. The IntaWeb forms added to the project and the controls on it will be rendered as HTML content and served to the browser.

Building FormData Application in Delphi

From the Delphi menu, choose File|New|Other... .In the New Items window, choose the IntaWeb tab. To create a new application, select the IntaWeb Application Wizard icon and then click OK:

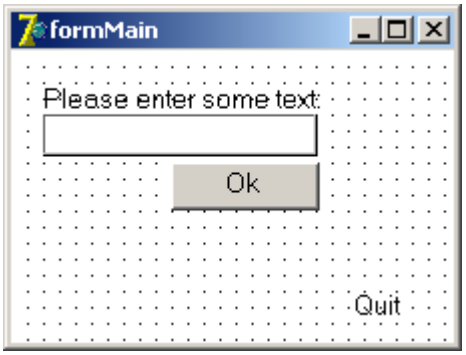


You will be prompted with an window which allows selection of different types of applications. Choose Stand Alone Application:

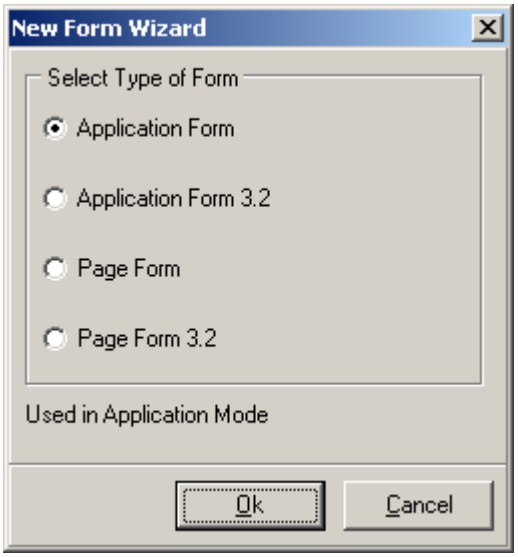


After completing these steps, the new application will have the default configuration: a project file named Project1 and Unit1.pas and IWUnit.dfm, an empty form.
Change the name of the project to FormData and the name of the form to formMain. Now, on this form, add from the IW Standard palette, a TIWEdit, a TIWLabel, a TIWButton, and a TIWLink. At runtime, these controls will be rendered as standard HTML tags, so they can be interpreted by the browser as any HTML file. The TIWEdit component will be rendered as an INPUT tag with type='text',

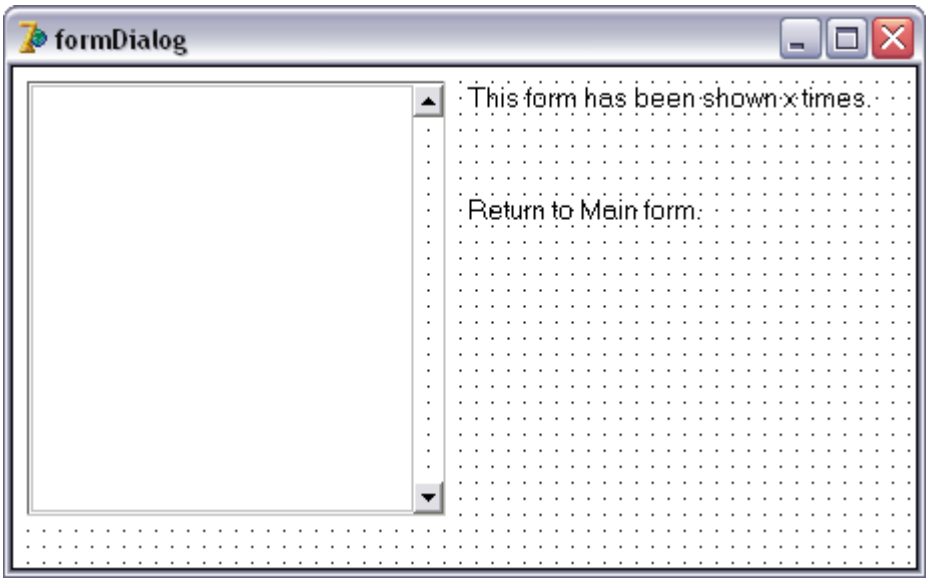
the TIWLabel component will be rendered as a SPAN tag, the TIWButton component will be an INPUT with type='button' and the TIWLink component will be rendered as an anchor tag.
Eventually, the form must look like this:



Next, add a new form to the project form the Delphi menu. Select File|New|Other... . In the New Items window, select New Form icon. The New Form Wizard will pop up:



Select the Application Form radio button and then press ok. Name the newly created form formDialog and add the following components on it: an TIWMemo, an TIWLabel and an TIWLink. Name the TIWLabel component lblCount and the TIWLink lnkReturn.
Change the Caption properties of the TIWLabel and the TIWLink control, so that the form looks like this:



At runtime, the controls will be rendered as HTML tags, as explained before. The TIWMemo component becomes an TEXTAREA tag.

Now, add a public member to the TformMain class of type TFormDialog:

```
FDialogForm: TformDialog;
```

And a public member to the TformDialog class:

```
FCount: Integer;
```

Next step is to add an event handler for the TIWButton component. For adding an event handler to an IntaWeb component, proceed as for any Delphi component: open the Object Inspector window and choose the Events tab. The TIWButton class has two published events:

- *OnClick* event: fired when the user clicks the button in the browser's window
- *OnHTMLTag* event: fired when IntaWeb builds up the tag which the control is rendered as. This event is used for adding optional attributes to the rendered tag.

In this demo, will write only the *OnClick* event handler. It will show the FDialogForm, the form which will display the number of times it has been shown and the text entered in the TIWEdit component of the main form.

This is the code that accomplishes this:

```
procedure TformMain.butnOkClick(Sender: TObject);
var
  s: string;
begin
  s := Trim(editText.Text);
  editText.Text := '';
  if s = '' then begin
    { navigate to the second form only if some text was entered }
    WebApplication.ShowMessage('Please enter some text.');
  end else begin
```

```
with FDialogForm do begin
    { add edited text to the memo }
    IWMemo1.Lines.Add(s);
    { increase the number of times the form was shown }
    Inc(FCount);
    { show the form }
    Show;

  end;
 end;
end;
```

Note that before calling the butnOnClick procedure, the FDialogForm must be initialized. This is done in the OnCreate event of the main form:

```
procedure TformMain.IWAppFormCreate(Sender: TObject);
begin
  FDialogForm := TformDialog.Create(WebApplication);
end;
```

Until now, the navigation between the TformMain and the TformDialog has been developed, and we have to write the code which displays the number of times the user has visited the second screen and the code which will redirect the user to the main form.

First, add the OnRender event handler for TformDialog. The OnRender event is used when users want to interact with the rendering process. The rendering process is the one who generates HTML, starting from the IntaWeb form and the controls on it. Use the same steps for adding this event handler as for the TIWButton component.

```
procedure TformDialog.IWAppFormRender(Sender: TObject);
begin
  lablCount.Caption := 'This form has been shown ' + IntToStr(FCount) + ' times.';
end;
```

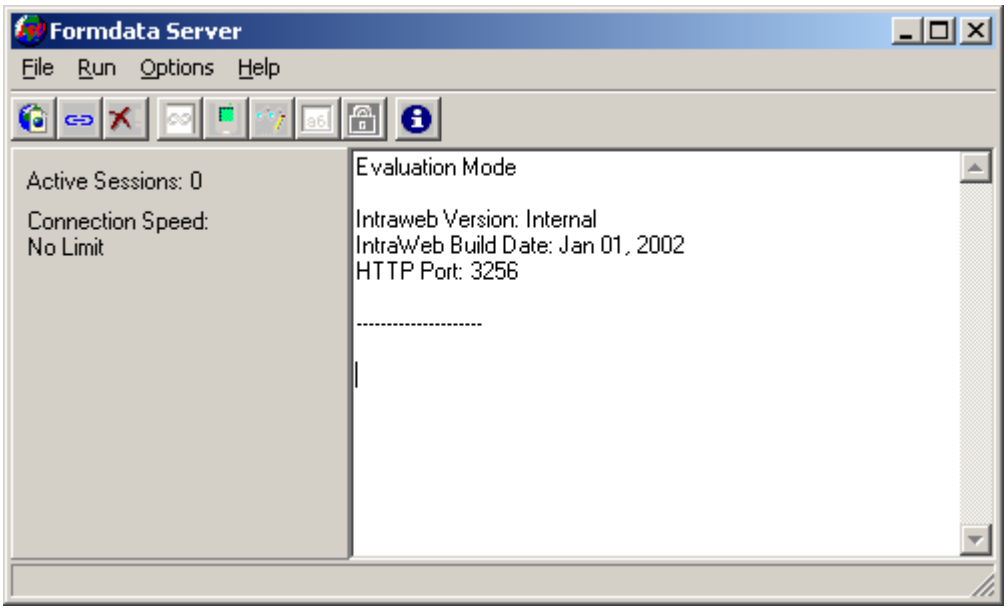
This code changes the Caption of lablCount.
Then add code for the OnClick event of lnkReturn.

```
procedure TformDialog.linkReturnClick(Sender: TObject);
begin
  Hide;
end;
```

Why does this code work? When wanting to show a form, you have to call it's Show method. For hiding a form, call it's Hide method. What happens with the forms in memory? They are kept in a stack, depending of the time they were created. When hiding the form from the top of the stack, the one created before it is shown.

Testing The Application

You can test your application like any other Delphi application. You will see a dialog similar to this:



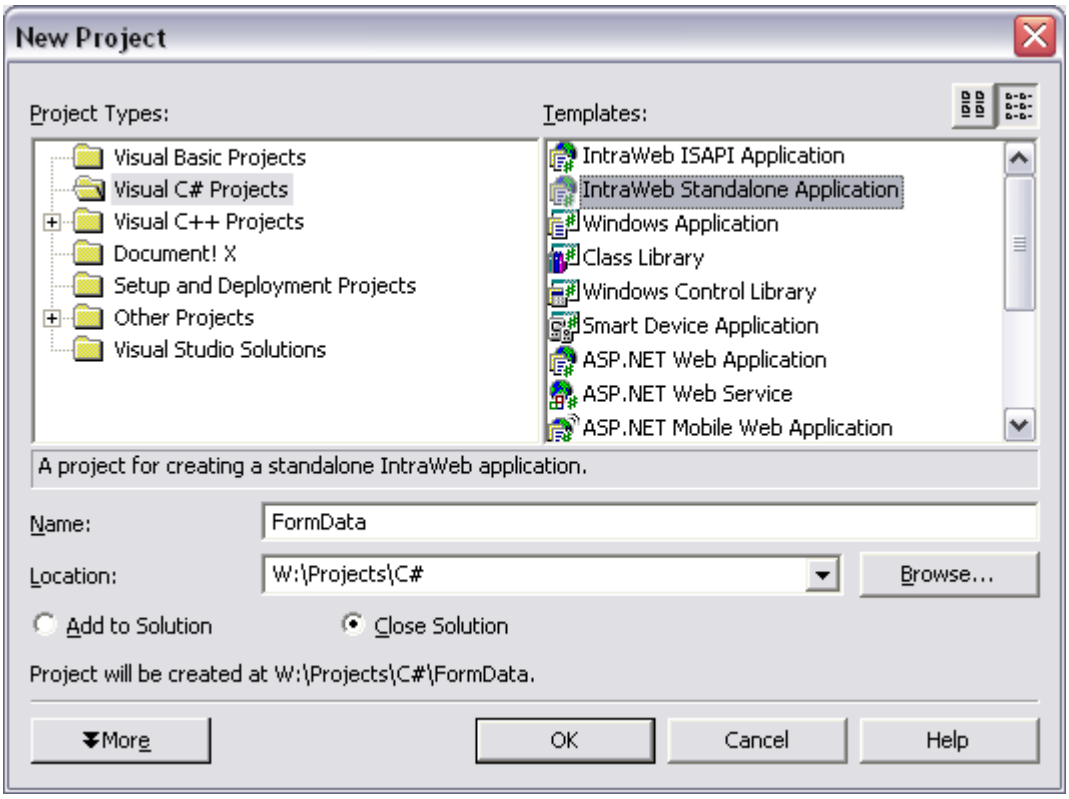
Press F9 to start testing your application.
You can turn on debug output by pressing the spectacles button on the toolbar. Then, in the left side of the window you will see the sessions IDs, calls to JavaScript functions, etc.

Deploying The Application

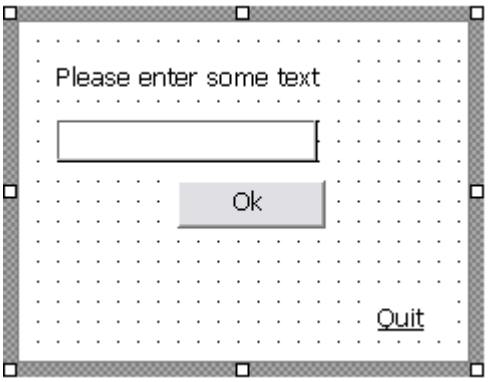
The application will be a self contained executable, and the only thing on the server side that you'll need to run your application. You do not even need a web server of any kind, because stand alone applications embed an web server.

Building FormData Application in Visual Studio .NET

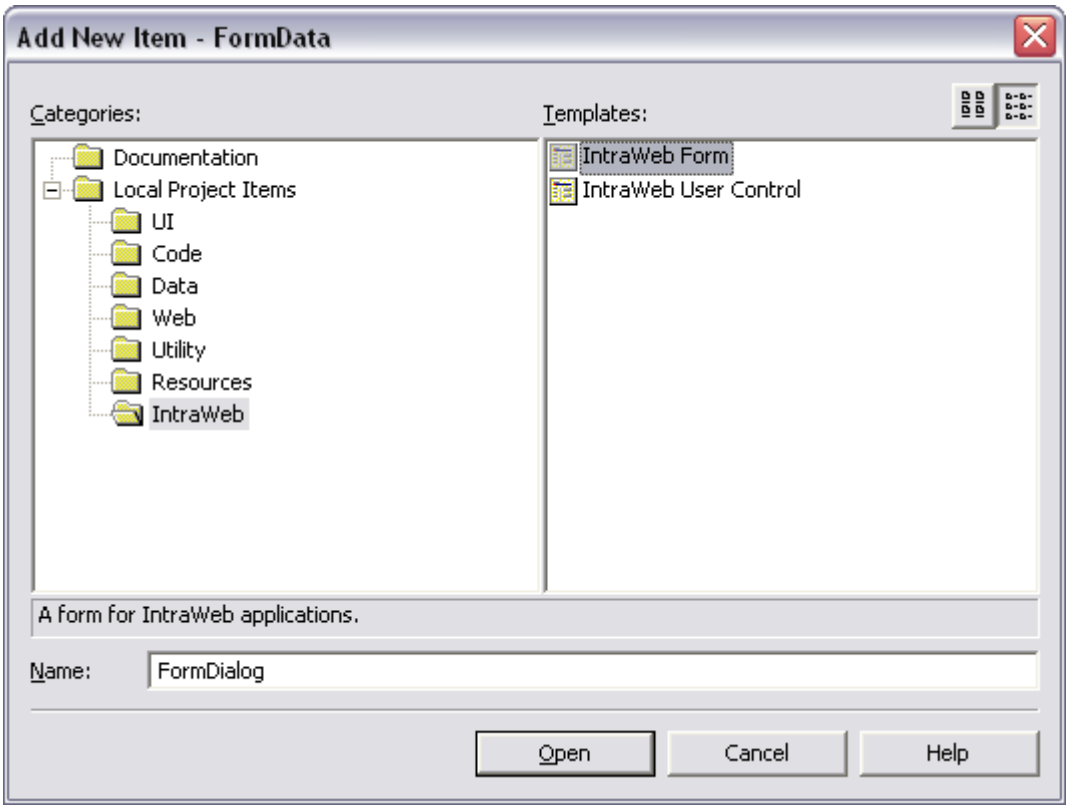
From the Visual Studio menu, choose File|New|Project... .In the New Project window, choose Visual C# Projects, and Intraweb Standalone Application.



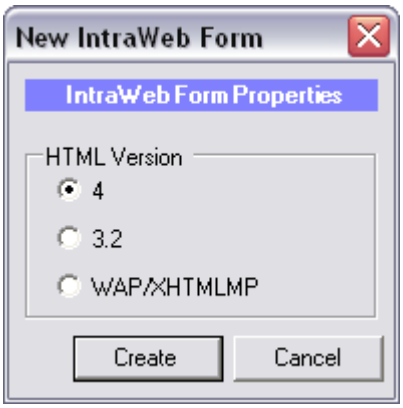
Choose a name for the new project and click OK. On the main form created add from the Intraweb Controls toolbox item an Edit, a TextLabel, a Button, and a Link. At runtime, these controls will be rendered as standard HTML tags, so they can be interpreted by the browser as any HTML file. The Edit component will be rendered as an INPUT tag with type='text', the TextLabel component will be rendered as a SPAN tag, the Button component will be an INPUT with type='button' and the Link component will be rendered as an anchor tag.
Eventually, the form must look like this:



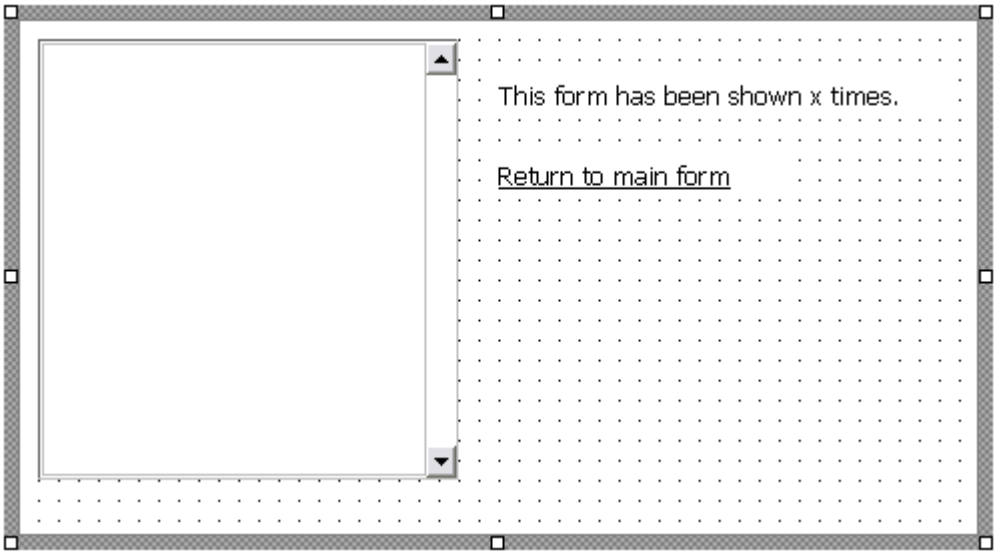
Next, add a new form to the project from the Visual Studio .NET menu. Select File|Add New Item... . In the Add New Item window, select Intraweb form.



The New Form Wizard will pop up:



Select the HTML 4.0 radio button and then press the button labeled create. Name the newly created form FormDialog and add the following components on it: a Memo, a TextLabel and an Link. Name the TextLabel component lblCount and the Link lnkReturn.
Change the Caption properties of the Label and the Link control, so that the form looks like this:



At runtime, the controls will be rendered as HTML tags, as explained before. The Memo component becomes an TEXTAREA tag.

Now, add a public member to the FormMain class of type FormDialog:

```
private FormDialog formDialog = new FormDialog();
```

And a public member to the TformDialog class:

```
public Int32 Count = 0;
```

Next step is to add an event handler for the Button component: in the properties tab for the Button control, select Events.
There will be two items for the Events tab:

- *OnClick* event: fired when the user clicks the button in the browser's window
- *OnHTMLTag* event: fired when IntaWeb builds up the tag which the control is rendered as. This event is used for adding optional attributes to the rendered tag.

In this demo, will write only the *OnClick* event handler. It will show the FDialogForm, the form which will display the number of times it has been shown and the text entered in the Edit component of the main form.

This is the code that accomplishes this:

```
private void btnOk_OnClick(object sender, System.EventArgs e)
{
        string s = "";
        if (editText.Text != null) {
                s = editText.Text.Trim();
        }
        if ("" == s) {
                WebApplication.ShowMessage("Please enter some text.");
        }
        else {
                formDialog.GetMemoControl().Lines.Add(s);
```

```
                formDialog.Count += 1;
                formDialog.Show();
        }
}
```

Until now, the navigation between the FormMain and the FormDialog has been developed, and we have to write the code which displays the number of times the user has visited the second screen and the code which will redirect the user to the main form.

First, add the OnRender event handler for FormDialog. The OnRender event is used when users want to interact with the rendering process. The rendering process is the one who generates the HTML displayed in the browser, starting from the IntaWeb form and the controls on it. Use the same steps for adding this event handler as for the Button component.

```
private void FormDialog_OnRender(object Sender)
{
        lblCount.Caption = String.Format("This form has been shown {0} times", Count);
}
```

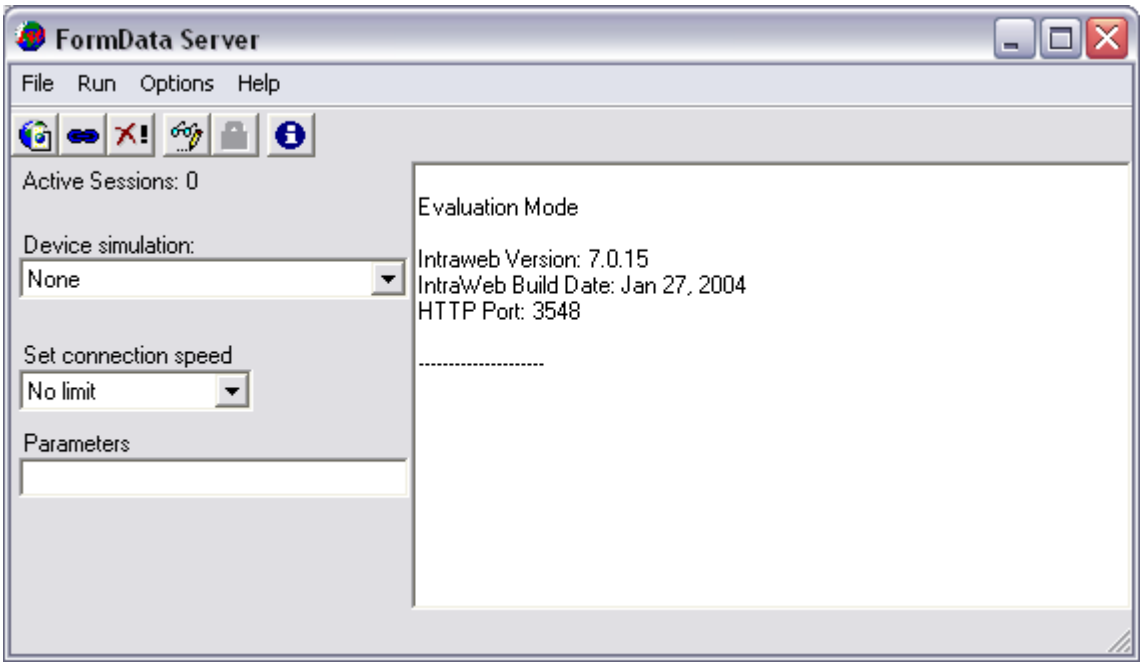
This code changes the Caption of lablCount.
Then add code for the OnClick event of lnkReturn.

```
private void lnkReturn_OnClick(object sender, System.EventArgs e)
{
        Hide();
}
```

Why does this code work? When wanting to show a form, you have to call it's Show method. For hiding a form, call it's Hide method. What happens with the forms in memory? They are kept in a stack, depending of the time they were created. When hiding the form from the top of the stack, the one created before it is shown.

Testing The Application

You can test your application like any other Visual Studio .NET application. You will see a dialog similar to this:



Press F9 to start testing your application.
You can turn on debug output by pressing the spectacles button on the toolbar. Then, in the left side of the window you will see the sessions IDs, calls to JavaScript functions, etc.

Deploying The Application

The application will be a self contained executable, which requires the NET framework 1.1.

5.3 Features

This demo is available only for Intraweb for Delphi.

The Features demo is not designed as a functional application. It is designed as a demo to show off specific features of IntaWeb. Use this demo to see how to use specific features in IntaWeb but also to see some of the things that IntaWeb can do.

5.4 Guess

Guess is a very simple application and contains only one form. It is however the ideal starter demo to understand how IntaWeb works. The demo serves as a simple guess the number game.

5.5 GuessWB

This demo is available only for Intraweb for Delphi.
GuessWB is the Guess application implemented using page mode instead of application mode.

5.6 GuessMulti

GuessMulti is a demo which simultaneously supports normal browsers as well as PDA's in a single executable.

5.7 FishFact

This demo is available only for Intraweb for Delphi.

FishFact is an IntaWeb port of the popular Delphi Demo FishFact. This demo requires DBDemos.

5.8 FishFactDM

This demo is available only for Intraweb for Delphi.

FishFactDM is the same as the FishFact demo, but demonstrates how to use Datamodules with IntaWeb. FishFactDM also shows how to present a paged grid. This demo requires DBDemos.

5.9 Phoenitics Customer Profiler

Phonetics Customer Profiler is a sample application that a cellular company might use to create custom proposals for potential customers and assist them in choosing which calling plan suits them best. The demo does not use a database, but it does demonstrate how to track information, use multiple forms, and templates.

5.10 StandAloneSSL

This demo is available only for Intraweb for Delphi.

StandAloneSSL demonstrates the use of stand alone mode with an SSL connection using sample certificates.

5.11 WebSnapSurvey

This demo is available only for Intraweb for Delphi.

WebSnap survey shows how to use IntaWeb page mode with WebSnap. This demo only works in Delphi 6 Enterprise and higher as WebSnap is new to Delphi 6 and only included in Delphi Enterprise.

5.12 Custom StandAlone

This demo is available only for Intraweb for Delphi.

The custom standalone demo demonstrates how to implement a standalone server with a custom interface using the IWStandAlone component.

5.13 Back Button

This demo is available only for Intraweb for Delphi.

The Back Button demo demonstrates how to use the OnBackButton event of the IWServerController to enable back-button functionality in the browser.

5.14 Page Forms

This demo is available only for Intraweb for Delphi.

PageForms is a simple demo that shows you how to implement PageMode application using WebBroker. For more information see the PageMode section in the manual

5.15 FishMarket

This demo is available only for Intraweb for Delphi.

FishMarket demonstrates how to use Re-Entry command in combination with cookie tracking to allow you to exit and re-enter an IW application by passing parameters on the URL.

5.16 Die, Fly ! Die !

Die, Fly ! Die ! is a complex demo that demonstrates both basic and advanced techniques to use with IntaWeb.
The demo is a virtual shop, when users can buy various fly killing products, ranging from a fly swatter to a nuclear bomb.

This demo is a replica of the well known ASPX demo IBuySpy from Microsoft, and it shows how easily complex tasks can be done with IntaWeb, without writing a single line of HTML or JavaScript code.

The demo makes heavy use of databases, and for this purpose the DBISAM engine from Elevate Software has been chosen for the Delphi implementation and MSDE for the .NET version. In order to compile and run the Delphi demo, you will need to obtain a copy of DBISAM. An evaluation version as well as the full version can be obtained from their website at http://www.elevatesoftware.com. To get the best results, please make sure you are downloading the latest engine version. For running the Visual Studio .NET 2003 demo, you need to install the MSDE on your machine and restore the database from the backup file delivered with the demo.

Why DBISAM ?

Long discussions took place before choosing this engine. The choice is base on these facts:

- it's fast
- it has a small footprint
- it compiles into the executable or dll and does not require additional components
- is fully SQL 99 compliant
- installation takes very little time

Source Code

Source code of Die, Fly ! Die ! has been written with the purpose of serving as a programming tutorial for IntaWeb users. Browsing the code you can find out how to use frames, how to use the OnHTMLTag, how to program a shopping cart, how to store images in a database and retrieve them

to a form and other tips and tricks users have requested over time.

5.17 WebMail32

WebMail32 is a powerful demo that show how easy you can get a complex PDA application using IntaWeb.
This demo simulates a simplistic web mailer that renders its output to HTML 3.2.

WebMail32 demonstrates the use of several HTML 3.2 controls, as well as the use of static and dynamic templates with IntaWeb and dynamic component creation.

Indy
WebMail32 uses the TIdPOP3 component, that is part of the Indy Component Library. Indy is an open source component library for Internet protocols and connections and comes bundled with Delphi 6 and Delphi 7.

You can also get the latest version of Indy for free from http://www.indyproject.org.

Section

6 Debugging

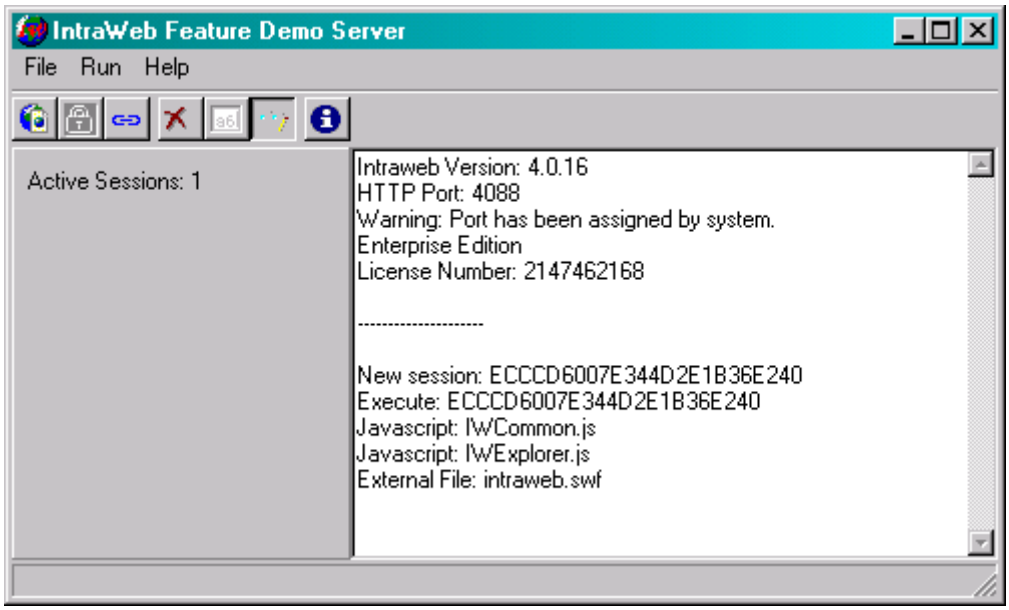
6.1 Getting Started

When using standalone mode debugging is the same as any other Delphi or Visual Studio .NET application. Just set your break points, watches, etc. and run.

6.2 Debug Output

While running in stand alone mode you can turn on debug output to see sessions created, destroyed, and HTTP requests. You can turn on debug output you can select "Show Debug Information" from the file menu, or depress the tool bar button that has an icon of the spectacles.

This is a screen shot with the debug tool bar button depressed, and debug output information from one user session:



6.3 Detecting Errors on Startup

If errors occur during start up of an application IntaWeb will terminate the application and log the error to an .err file. The applications filename with an .err extension will be used. If you are having trouble starting an application, check for an associated .err file. The .err file is a text file and can be viewed with notepad, or edln.
Errors that occur outside of the program block such as missing required packages or statically linked DLLs cannot be detected and will not be logged in the .err file.

6.4 Command Line Parameters

Auto Browser Launch

To further expedite development you can add "/LaunchBrowser" to the application parameters to have the program launch the browser automatically each time it is run. To do this in Delphi, select the Parameters menu item from the Run menu. Then enter "/LaunchBrowser" in the Parameters field. The next time you execute your application the browser will automatically be launched. Same thing can be achieved in Visual Studio .NET by entering "/LaunchBrowser" for the Command Line Arguments

option from the project's properties pages, Configuration Properties/Debugging section.

Auto Minimize

The debug screen can be told to start minimized by passing /minimize on the command line. This is useful during development if combined with /LaunchBrowser

Section VII

7 Development

7.1 Rethinking the User Interface

Many people try to design their web applications exactly like normal applications. If you try this, you will create interfaces that do not work well. Imagine making a windows application behave like a DOS application (WordPerfect did this with their initial Windows port). That would be an awkward interface would it not? Not only do you need to think differently about your user interface for the web, you also need to realize that the web has limitations and design around them.

One example of this is DBGrids. In a normal Delphi application, it might be considered normal to display hundreds or thousands of records in a grid. Doing such on the web will create very large HTML documents and very slow load times for the user.

Once developers realize this fact, they often ask for "Next" and "Previous" buttons and that the DBGrid be expanded to allow partial display. While this could be implemented, it would need to be implemented either to consume large amounts of memory on the server, or by constantly requiring the database which would consume less memory but would be slow. Instead of approaching it like a normal Delphi application, rethink your interface for the web.

Certainly not the only possibility, but a common one is the following technique. Instead of presenting your users with thousands of records initially, present them with a blank grid and a search field. Require your users to present some basic criteria to locate the records that they need. Using the search criteria, you can then return dozens, or just a few hundred rows. Not only is this good for bandwidth, but it is a good user interface, and will minimize the load on your database.

Allowing users to enter search criteria still allows for the possibility that the results may still number in the thousands and cause the very same problem that you were trying to avoid. To assist with this, TIWDBGrid has a RowLimit property. It defaults to 0, which means it is disabled. You can set it to a maximum value, and no matter how may rows the query returns, no more than the number in RowLimit will be returned to the user.

If you think about this, you have probably seen this technique elsewhere. Many search engines limit the number of rows that are returned. This is not only for bandwidth reasons. In most cases, the data becomes diminishingly useful after a certain number. In cases where this is not true, simply too much data is given to the user at one time and they will likely filter it anyways.

If you still decide that you do want a "paged grid" consisting of small sets of data with next / previous options you can accomplish this by setting the TIWDBGrid's StartFirst property to false and setting the RowLimit property to the number of rows you wish to display at a given time. Then by positioning the dataset before display, you can move next / previous.

7.2 Writing Your First IntraWeb Application

Using Delphi

All IntaWeb applications should be created using the IntaWeb Application Wizard in the IntaWeb tab in the repository. Click on File -> New -> Other and then choose the IntaWeb tab. Select the IntaWeb Application Wizard. Finally click Ok.

This creates a framework for a new IntaWeb stand alone application. Although the project can be compiled and executed at this stage, it does not do anything. The standard debug form comes up displaying some information about IW and menu items to debug the application. Selecting the Execute (or pressing the F9 key) menu item will launch the browser with a blank page. This is because main

form does not contain any components or functionality yet.

```
program Project1;

uses
  Forms,
  IWMain,
  ServerController in 'ServerController.pas' {IWServerController: TIWServerController},
  Unit1 in 'Unit1.pas' {IWForm1: TIWFormModuleBase};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TFormIWMain, formIWMain);
  Application.Run;
end.
```

The code in the figure above displays the contents of the project file. You can see that it is the same as a standard Delphi application. This is true for stand alone applications.

As mentioned previously, the new project is the basic building block for any IntaWeb application. Like any other Delphi project, a main IntaWeb form is created and can be used as the main form of the application. To demonstrate the power and the facility of IW applications, below, a small example is shown.

1. Open up the default IWForm (IWUnit1.pas) that has been created.
2. Drop a TIWButton, TIWEdit and TIWLabel on the form in no particular order.
3. Assign the following code to the Button's OnClick event:

```
procedure TIWFormModule.IWButton1Click(Sender: TObject);
begin
  IWLabel1.Caption := IWEdit1.Text;
end;
```

Once the steps are complete, compile and run the application. To test it, press the F9 key. The default browser should be launched and display the main form. Enter some text in the text box and click on the button. The output is displayed in the label.

Using Visual Studio .NET

To build an IntraWeb.NET application choose IntaWeb Standalone Application or IntaWeb ISAPI Application form the New Project window in Visual Studio .NET. Select the default options in the wizard and check the code generated by the wizard for the application's entry point:

```
[STAThread]
static void Main()
{
        IWServerController.SetServerControllerClass(typeof(IWServerController));
        FormMain.SetAsMainForm(typeof(FormMain));
        Application.Run(new Main());
}
```

If you click Start at this stage, and launch a browser form the Debug Window, the page shown in the browser will be blank, since no control was yet placed on the form and no extra code written.
Try this simple example:

1) On the formMain, drop a Button, a TextLabel and an Edit control.
2) Assign the OnClick event of the button to this function

```
private void button1_OnClick(object sender, System.EventArgs e)
```

```
{
        textLabel1.Caption = edit1.Text;
}
```

Now compile and run the application. Click Start and press F9 in the Debug Window. The default browser will be launched. Enter some text in the edit and click the button.

Granted what we have shown in these examples is not rocket science. But, it has been created using standard Delphi/C# code and without any HTML. The example presented has been chosen because of its simplicity. The purpose is to demonstrate that programming IntaWeb applications is very much the same as developing any other windows application. The same methods, properties, events, etc. can be used in the same way. However, IntaWeb is much more powerful and can be used to create fully enabled database applications and more.

For a more detailed introduction to creating new applications and detailed tutorials, please see tutorials on the Atozed website at http://www.atozed.com.

7.3 Images and Graphics

IntaWeb supports graphics via the use of templates, TIWImage, TIWDBImage and TIWImageFile Delphi components or the equivalents WebImage, ImageFile and DBImage controls in Visual Studio .NET. There are many ways to use graphics with in IntaWeb, but these are the primary methods.

Templates

Use of graphics in templates is done by inserting the graphics directly into the HTML. Graphics may be served using the Files directory, or a standard web server.

TIWImage and WebImage

TIWImage (WebImage in Intraweb for Visual Studio .NET) is used for dynamic images. Each time an image is requested the image is converted to a JPG. This can be rather resource intensive and thus should only be used for images that will be changed as part of an application's function.

For an example of this, please see the Dynamic Interactive Image demonstration in the Features demo.

For static images that are not generated each time, use TIWImageFile.

TIWDBImage and DBImage

TIWDBImage (DBImage in Intraweb for Visual Studio .NET) converts images from a database field to a JPG automatically. It is used just like a normal TDBImage, it performs all the work necessary to display the image from the database field into the browser.
For an example of TIWDBImage, see either of the FishFact demos.

TIWImageFile and ImageFile

TIWImageFile (ImageFile in Intraweb for Visual Studio .NET) serves a file directly from a file on disk. Because it does no conversion of the image, TIWImageFile is an extremely efficient way to serve images and is much more efficient than TIWImage. If you are using images that are completely static, you should always use TIWImageFile.

TIWImageFile provides for design time support as well by displaying the image at design time. However the image is merely displayed, the image data is not stored with the form. Whenever displayed at design time the image is loaded from the file on disk.
The filename specifies a full path and filename to the image file to display at design time.
At run time, the path is ignored and only the filename is used. At run time, the image is expected to be

in the files directory.

GIF Support
IntaWeb for Delphi can support GIF files however the install does not install GIF support. Please see the IntaWeb FAQ for details on how to use GIF files with IntaWeb. IntaWeb for Visual Studio .NET has GIF support.

7.4 Extending IntraWeb

Custom Components
All IntaWeb components are written using an open API that easily allows you to write your own components and add them to IntaWeb just as you can with Delphi. To further facilitate the writing of components the source code for all IntaWeb components in included, even in the evaluation edition of IntaWeb.

For further information on creating custom components please see the section "Writing Custom Components" in this manual.

Embedding Raw HTML
You can also embed your own HTML in your IntaWeb application without writing a component by using the TIWText component. Simply drop a TIWText component on your form. Set the RawText property to True, and the WantReturns to False. Finally, put the HTML in the Lines property and your custom HTML will be output as part of your form.

HTML Templates
HTML templates (simply referred to as templates elsewhere) can be used to add advanced HTML into your application and customize the look of your application. Please see the section on layout managers for more details.

7.5 Working with COM

Initialization of COM

IntaWeb can automatically make the necessary calls to initialize COM. This is very much useful when working with components such as ADO to access data. To use this feature, set the property ComInitialization in the ServerController. It can take one of three possible values:

- **ciNone**: No COM intialization is called. This is the default.
- **ciNormal**: COM is initialized. This is the method used when deploying Standalone applications built with IntaWeb. It specifies the concurrency model as single-thread apartment.Internally, IntaWeb will make a call to CoInitialize(nil).
- **ciMultiThreaded**: Use this setting when requiring COM initialization and when deploying as an ISAPI DLL.This will initialize COM for multi-threaded object concurrency. Internally, IntaWeb will make a call to CoInitializeEx(nil, COINIT_MULTITHREADED).

Uninitialization is also taken care of automatically when the property is set to something other than ciNone.

Threads and COM

Most COM objects cannot be passed between threads. IntaWeb application are multi threaded HTTP servers that reuse threads, therefore you cannot store references to COM objects between two accesses to the same session or between sessions. Depending on your problem specific, you can

either use a COM object in a single operation (not across the whole session, but at one access at a time), then discard it, or marshal the COM interfaces needed by yourselves.

7.6 Working with ClientDataSet Components

This chapter refers only to IntaWeb for Delphi.

If you are using a ClientDataSet component and you get an Access Violation when exiting the application, you need to add the *DBClient* unit in the application uses clause **before** *IWMain.*

The reason for the access violation is that if *DBClient* is not included in project file uses clause, it's internal interfaces are freed before all sessions are closed and when IW closes it's sessions it will try to free ClientDataSet component, and you will get the access violation.

When *DBClient* is placed before *IWMain* IW will free sessions before *DBClient* interfaces are freed.

7.7 Working with PDA

IntaWeb 5.1 introduces support for PDA. This is accomplished by restricting the output generated by IW application using pure HTML 3.2 without any additional JavaScript or Cascading Style Sheets. Although at first this would seem that application development is limited, it is important to understand that the limitations are mostly imposed by the devices rather than IW. Since HTML 3.2 is a standard that nearly all devices on the market support, it is a good option for developing web applications focused on PDA's.

Developing 3.2 applications
Developing in Delphi
When developing PDA applications, you need to use Applications forms (alternatively Page forms) that are of the class TIWAppForm32 (TIWPageForm32). This can be accomplished either using the Wizard to create a new application and selecting "Main Form 3.2" or adding a 3.2 form to an existing application.
Developing in Visual Studio .NET
Use the wizard to create a new application and make sure that all the forms added to the application to be HTML 3.2 compliant (inherited from AppForm32 class).

Using 3.2 controls
PDA applications are restricted to controls that are of the class 3.2. However, due to the restrictions imposed by HTML 3.2, these counterpart controls sometimes do not have all the functionality that the 4.0 (or normal) IW controls have. You can ONLY use 3.2 controls for PDA applications.
Using Intraweb for Delphi 3.2 controls
There are two tabs on the component palette that have the corresponding 3.2 version component of the standard IW components.
Using Intraweb for .NET 3.2 controls
Visual Studio .NET automatically displays only 3.2 components in the toolbox when a 3.2 form is active.

Rendering and layout

By default, when placing 3.2 controls on a form, these are rendered in vertically in tab order. This is ideal for testing your application before finalizing the UI aspects of it. The reason for the vertical placement is due to the fact that PDA's do not support CSS and therefore positioning cannot be accomplished directly. To allow correct positioning, you have two alternatives. The first is to use a TIWLayoutMgrHTML32 component and design your form using the editor. The second option is to use a TIWTemplateProcessor32 (TemplateProcessor32 in Intraweb for Visual Studio .NET) and use templates to design the look of the application. However, it is important to remember that when using an external editor to design 3.2 forms, you need to restrict the HTML elements to 3.2 standard. Most HTML editors allow you to set this option before editing.
Note: In Intraweb for Visual Studio .NET, the LayoutMgrHTML32 editor can not be used for designing the form.

Setting the main form
As with normal IW applications, PDA versions need to have a main form defined. To do this in Intraweb for Delphi, use the SetAsMainForm method in the initialization section of the form you require to be the main form. In Intraweb for Visual Studio .NET declare the Main method in the form wanted as main form.

Running the application
During development, it is much easier to test PDA applications using the computer you develop on. IW allows you to do this by launching your normal browser and testing the PDA version. To do so, you have two alternatives:

1. Using the Standalone application, click on the PDA icon before launching the browser with the Launch Browser button.
2. Directly type the URL in the browser appending 32 at the end, e.g. http://127.0.0.1:8888/32

This will allow you to see the results of your IW PDA application on the browser.

When using the application directly from a PDA device, IW will detect the browser that the PDA device uses and automatically launch the PDA application.

Note: The error message "no main form is defined" occurs when you try to access an application that doesn't have HTML 3.2 forms with a browser that it's not able to display HTML 4.0. This might occur with Netscape 4, as Netscape 4 doesn't have full support for HTML 4.0 and JavaScript.

7.8 Miscellaneous

External Files
Files such as images and download can be accessed using relative paths located under the main application folder. Create a folder named files and place all HTML objects referenced inside it. In the HTML page you can reference the images using:

```
img src="../files/image.jpg"
```

Be sure to use / and not \. Internet Explorer will correct for \, but other browsers will show broken images. In addition, this functionality is not limited to images and can be used for any file type.

Files accessed with the files URL are cached by the browser. If you wish to create dynamic files that should not be cached use ../filesnc/<filename> instead of ../files/<filename>. Files will still be retrieved from the same place in the files subdirectory, but the browser will be instructed not to cache them.

Other Form Properties
Be sure to look at the properties on the form as well. There are properties that allow customization of

the output that are often overlooked. These properties allow control over the HTML output, and more.

Server Controller

Each application has a ServerController unit. The ServerController contains properties to affect how the application acts and behaves on a global scale. It also contains events that can be defined. For more information, see the Server Controller chapter.

Datamodules

If you use datamodules, please see the FishfactDM demo. One thing to note, if you link your datasource properties to a datamodule at design time like FishFactDM does, your datamodules MUST be owned by the users WebApplication. This is done in FishFactDM by setting the datamodule's owner to the session data's owner, which is the WebApplication variable. If this is not done, the forms will not be read in properly and all the forms will be linked to the first and same datamodule.
Note: The Datamodules refers only to Intraweb for Delphi.

Section VIII

8 Form Management

8.1 Working with Forms

All forms used by an IntaWeb application must be an IntaWeb-specific form. Working with IntaWeb forms differs a little bit from working with standard Delphi forms or Windows forms in Visual Studio .NET. For instance, any form that is displayed must be done using the Show method of the form. In other words modal show is not permitted or supported.

In Intraweb for Delphi
The repository contains another icon, which is the IntaWeb Form. All new IntaWeb forms should be created using File | New and choosing the IntaWeb Form. Standard Delphi forms are not compatible with IntaWeb. A new unit and form will be created and displayed on the screen.

In Intraweb for Visual Studio .NET
All new Intraweb forms should be created by selecting IntaWeb form in the Add Item dialog in Visual Studio IDE.

8.2 Update Mode

Starting with version 6, IntaWeb uses a revolutionary update model that will greatly improve your application's performance. This is called partial updates.

Application forms have now a new property, called UpdateMode. Setting this property to *umAll* will force IntaWeb to use the standard update mode, the traditional mode when the whole form is refreshed. Setting this property to *umPartial* will cause the form to refresh ony the controls that need refreshing, this improving significantly the speed of your application, especially over slower connections.

Partial update is the most innovative technology in web development. However, due to the heavy use of scripting technologies, this update mode is available only on latest browsers and only in HTML 4.0 mode.

Limitations of partial updates

When using partial updates, several things must be taken into the consideration.
Here is a list of the situations in which partial updates won't work properly:

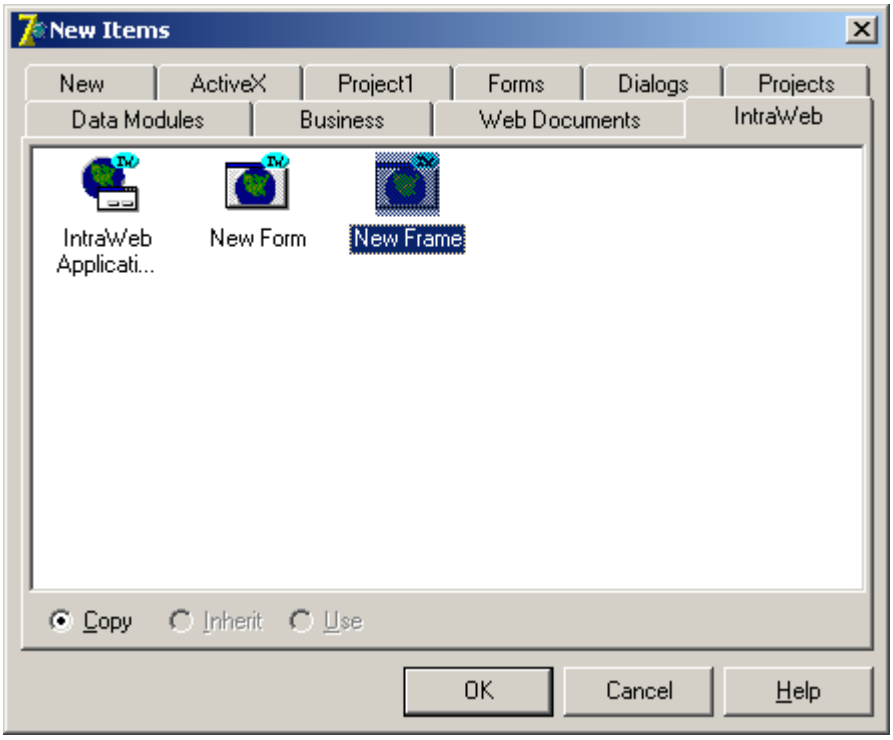
- TerminateAndRedirect(AURL, AMsg) won't work correctly if UpdateMode is umPartial
- Dynamic component creation won't work if UpdateMode is umPartial. To use dynamic component creation you'll have to use the umAll update mode
- Changing the size of the controls at runtime doesn't work currently when UpdateMode is umPartial. This is a known issue and will be corrected in a future version, but for the time being setting any size of a control (width, height etc.) at runtime does not work.
- The Treeview control won't work with umPartial update mode
- runtime assignment of event handlers won't work in umPartial

All of those situation are happening because of the complex JavaScript coding partial updates use. Some of these issues will be addressed in future versions of the product.

8.3 IntraWeb Frame

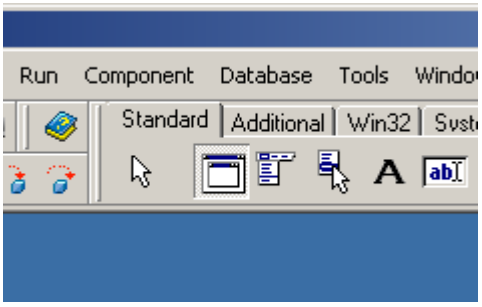
For using Delphi frames in your IntaWeb project, you must use the Frame Wizard which will create an IntaWeb frame, inherited form Delphi's TFrame.

First, from Delphi's main menu select File | New | Other... .
In the New Items window, choose the IntaWeb tab and select New Frame:



This operation will create an control inherited from TFrame, with a TIWRegion control placed on it. All components added to this frame must be placed on this TIWRegion component.

For adding the newly created frame on one of your forms, go to the Standard tab from Delphi's component palette, and choose Frames:

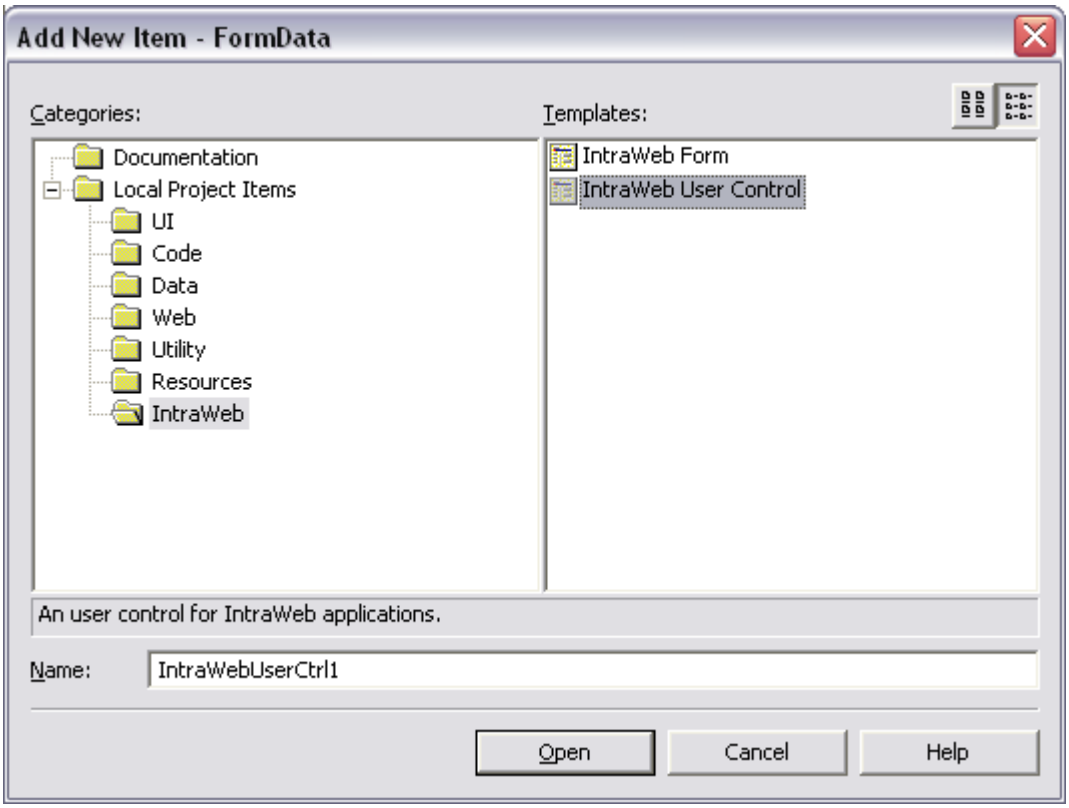


From now one, you can use this frame as any of Delphi's TFrame.

An example of frames usage with IntaWeb can be seen in the Features demo.

8.4 Intraweb User Control

The equivalent for Visual Studio .NET for Delphi Intraweb frames are the user controls. This type of controls act as a container for other Intraweb controls and can be placed on various forms. User controls can be built in Visual Studio .NET using the wizard: select File|Add New Items from the IDE's menu and choose from the Intraweb section Intraweb user control:



The newly created control will be automatically placed on the "My IW User Controls" tab in the toolbox. Note that when a change occurs in an user control, you have to build the project in order for the changes to be taken into account by the forms hosting the control.

8.5 Visual Form Inheritance

Visual Form Inheritance is not supported, but code inheritance is. Visual Form Inheritance is problematic in normal Delphi applications and requires extra streaming support. In addition, Visual Form Inheritance does not work properly with custom forms which IntaWeb uses. Because of these reasons, no extra streaming support has been included in IntraWeb's code.

You can however use code inheritance. Code inheritance allows inheritance of methods, members and properties. You can also create controls dynamically.

8.6 Managing Forms

Form management in an IntaWeb application is very similar to that of a normal Windows application but with a few restrictions:

1. Only one form may be visible at any time. This is because the form is actually shown in the

browser.
2. Modal forms may not be used, however since only one form at a time may be visible, essentially all forms are modal.
3. In Intraweb for Delphi, forms must be owned by WebApplication.

8.7 Form List

IntaWeb keeps a list of forms as part of the users session. It is kept in a stack like fashion with newly shown forms being added to the top. When forms are hidden or released this list is used to determine the form that should be activated if not explicitly instructed to show another form via a call to the .Show method of a form.

Normally the form list is never directly interacted with by the user but instead methods of the forms are called. However there are cases where direct interaction with the form list may be necessary. For these cases TIWApplication (IWApplication in Intraweb for Visual Studio .NET) contains several methods for interacting with the form list and are documented in the help file.

8.8 Showing Forms

Intraweb Forms are shown by calling the Show method. One thing that is different however from a normal application is that the form is not shown immediately when the .Show method is called. With an IntaWeb application the call to the .Show method merely sets the form as the next active form to show after the event exits and returns to IntaWeb. Only after the event executes will the form be shown.

A given instance of a form can be shown multiple times to bring it to the top. In this case the instance of the form will be in the form list in multiple places.

Showing Forms in IntaWeb for Delphi

The general format to display a form is this:

```
TFormType.Create(WebApplication).Show;
```

This may be confusing at first, but it is just short hand for:

```
with TFormType.Create(WebApplication) do begin
  Show;
end;
```

This should be familiar to you as it is the same as in a standard application except the owner here is WebApplication.

Showing Forms in Intraweb for Visual studio .NET

The general form is:

```
(new FormType()).Show();
```

8.9 Hiding Forms

In a normal application a form can be hidden without destroying the form by calling the .Hide method.
The same functionality can be implemented in IntaWeb by calling its simply calling the .Hide method. The .Hide method will hide the form without destroying it as .Release does.

.Hide removes all references in the form list as .Release does but does not destroy the form. Because of this you must keep a reference to the form if you wish to redisplay it later, otherwise the form will become orphaned.

8.10 Destroying Forms

In a normal Delphi application when a form is no longer needed it can be destroyed using the .Free or the .Destroy methods.

In an IntaWeb application it is similar, however you must not call the .Free or .Destroy methods directly. Instead you must call the .Release method. The .Release method does not actually destroy the form when called. The form will not be destroyed until the event exits and returns control to IntaWeb. This is because .Release is usually called from within and event of the form itself, although this is not always the case.

After release is called, just like in a normal application the active form becomes the one that was active prior to the destroyed form became active. If you do not wish to return the user to the prior form you must call the .Show method for a different form.
The .Show method can be called before or after .Release since neither takes effect until control is returned back to IntaWeb.
When a form is released, all references to it in the form list are removed. This causes an alteration in the order of the forms that will be shown when forms are hidden or released with no explicit .Show method calls.

In Visual Studio .NET the forms are not explicitly deleted because of the garbage collector. Calling .Release is needed though, so the form is removed from the form list.

8.11 Passing Data Between Forms

Data can be passed between form just like in any normal application. Since forms are persistent information can be stored in member variables of form classes.
For demonstration purposes we will build a simple demo with two forms. The main form will contain a button and an edit box. The other form contains a memo field and a label. When the user presses the button on the main form the text from the edit box will be added to the memo on the second form and the form will be displayed. The second form will also display how many times it has been displayed and allow the user to return to the main form.

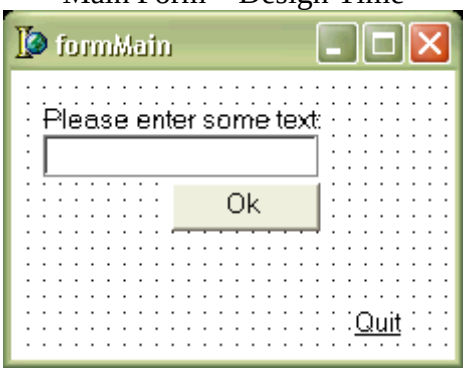
Complete Demo

To see the project in action, please see the FormData.dpr project in the Demos directory of IntaWeb for Delphi, or the FormData.sln in the Demos directory of Intraweb for Visual Studio .NET. For the Visual Studio .NET version, the demo is available in both C# and

VB.NET languages.

Delphi example
This is how the main form looks like at design time:

Main Form – Design Time



Main Form – Source Code

```
unit Main;
{PUBDIST}

interface

uses
  IWAppForm, IWApplication, IWTypes, IWCompButton, IWCompEdit,
Classes,
  Controls, IWControl, IWCompLabel, Dialog, IWHTMLControls;

type
  TformMain = class(TIWAppForm)
    IWLabel1: TIWLabel;
    editText: TIWEdit;
    butnOk: TIWButton;
    IWLink1: TIWLink;
    procedure butnOkClick(Sender: TObject);
    procedure IWAppFormCreate(Sender: TObject);
    procedure IWLink1Click(Sender: TObject);
  public
    FDialogForm: TformDialog;
  end;

implementation
{$R *.dfm}

uses
  SysUtils;

procedure TformMain.butnOkClick(Sender: TObject);
var
  s: string;
begin
```

```
s := Trim(editText.Text);
  editText.Text := '';
  if s = '' then begin
    WebApplication.ShowMessage('Please enter some text.');
  end else begin
    with FDialogForm do begin
      IWMemo1.Lines.Add(s);
      Inc(FCount);
      Show;
    end;
  end;
end;

procedure TformMain.IWAppFormCreate(Sender: TObject);
begin
  FDialogForm := TformDialog.Create(WebApplication);
end;

procedure TformMain.IWLink1Click(Sender: TObject);
begin
  WebApplication.Terminate('Good bye!');
end;

end.
```

IWLink1 OnClick

This event is hooked to the link with the caption "Quit" and simply terminates the user session when the user clicks the link.

OnCreate

The OnCreate event is called when the form is created. In this event another form is created and the reference to it is stored as a member variable of this form so it can be accessed again later.

butnOk.OnClick

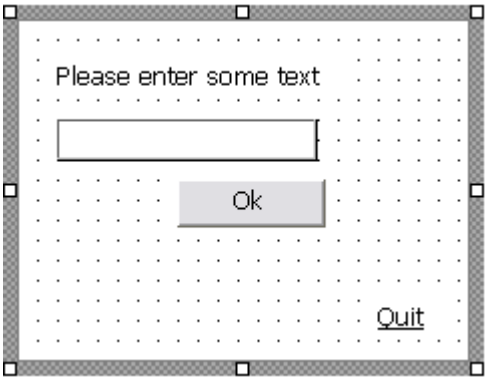
In the OnClick event the edit box is checked for data. If no data exists WebApplication.ShowMessage is called to display a message to the user. After the message is dismissed the main form is shown again.

If the user did enter data, using FDialogForm (which was created in this form's OnCreate) is used. Data is added to the memo, and a member variable of TFormDialog is updated. It is then displayed using the .Show method. As you can see, data is very easy to pass between forms and is the same as in a normal Delphi application.

Visual Studio .NET example

The forms are called: FormMain and FormDialog

FormMain - Design time



FormMain - C# source code:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;
using System.Data;
using Atozed.Intraweb.NETMain;
using Atozed.Intraweb.AppFormUnit;

namespace FormData
{

	public class FormMain : AppForm
	{
		private Atozed.Intraweb.CompLabel.TextLabel tiwLabel1;
		private Atozed.Intraweb.CompEdit.Edit editText;
		private Atozed.Intraweb.CompButton.Button btnOk;
		private Atozed.Intraweb.HTMLControls.Link lnkQuit;
		private System.ComponentModel.Container components = null;
		private FormDialog formDialog = new FormDialog();

		public FormMain()
		{
			// This call is required by the Windows Form Designer.
			InitializeComponent();

			// TODO: Add any initialization after the InitializeComponent call
		}

		/// <summary>
		/// Clean up any resources being used.
		/// </summary>
		protected override void Dispose(bool disposing)
		{
			if (disposing)
			{
				if (components != null)
				{
					components.Dispose();
```

```
                        }
                    }
                    base.Dispose(disposing);
                }

                #region Designer generated code
                /// <summary>
                /// Required method for Designer support - do not modify
                /// the contents of this method with the code editor.
                /// </summary>
                private void InitializeComponent()
                {

                }
                #endregion

                #region Application entry point
                /// <summary>
                /// The main entry point for the application.
                /// </summary>
                [STAThread]
                static void Main()
                {
                    IWServerController.SetServerControllerClass(typeof(IWServerController));
                    new IWLicenseKey();
                    FormMain.SetAsMainForm(typeof(FormMain));
                    Application.Run(new Main());
                }
                #endregion

                private void btnOk_OnClick(object sender, System.EventArgs e)
                {
                    string s = "";
                    if (editText.Text != null) {
                        s = editText.Text.Trim();
                    }
                    if ("" == s) {
                        WebApplication.ShowMessage("Please enter some text.");
                    }
                    else {
                        formDialog.GetMemoControl().Lines.Add(s);
                        formDialog.Count += 1;
                        formDialog.Show();
                    }
                }

                private void lnkQuit_OnClick(object sender, System.EventArgs e)
                {
                    WebApplication.Terminate("Good bye!");
                }

            }
        }
```

FormMain - VB.NET source code

```
Imports System
```

```
Imports System.Collections
Imports System.ComponentModel
Imports System.Drawing
Imports System.Windows.Forms
Imports System.Data
Imports Atozed.Intraweb.NETMain
Imports Atozed.Intraweb.AppFormUnit

Public Class FormMain
        Inherits AppForm
        Private WithEvents TiwLabel1 As Atozed.Intraweb.CompLabel.TextLabel
        Private WithEvents editText As Atozed.Intraweb.CompEdit.Edit
        Private components As System.ComponentModel.Container = Nothing
        Private frmDialog As FormDialog = New FormDialog


        Public Sub New()
                ' This call is required by the Windows Form Designer.
                InitializeComponent()

                ' TODO: Add any initialization after the InitializeComponent call
        End Sub

        ' <summary>
        ' Clean up any resources being used.
        ' </summary>
        Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
                If disposing Then
                        If Not components Is Nothing Then
                                components.Dispose()
                        End If
                End If
                MyBase.Dispose(disposing)
        End Sub

        #Region "Designer generated code"
        ' <summary>
        ' Required method for Designer support - do not modify
        ' the contents of this method with the code editor.
        ' </summary>
        Private WithEvents btnOk As Atozed.Intraweb.CompButton.Button
        Private WithEvents lnkQuit As Atozed.Intraweb.HTMLControls.Link
        Private Sub InitializeComponent()

        End Sub
        #End Region

        #Region "Application entry point"
        '/// <summary>
        '/// The main entry point for the application.
        '/// </summary>
        <STAThread()> _
        Shared Sub Main()
                Dim LLicense As IWLicenseKey = new IWLicenseKey()
                IWServerController.SetServerControllerClass(System.Type.GetType("FormData.IWS
erverController"))
                FormMain.SetAsMainForm(System.Type.GetType("FormData.FormMain"))
```

```
            Application.Run(new Main())
        End Sub
        #End Region

        Private Sub lnkQuit_OnClick(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles lnkQuit.OnClick
            WebApplication.Terminate("Good bye!")
        End Sub

        Private Sub btnOk_OnClick(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnOk.OnClick
            Dim s As String = ""
            If (editText.Text <> Nothing) Then
                s = editText.Text.Trim()
            End If
            If ("" = s) Then
                WebApplication.ShowMessage("Please enter some text.")
            Else
                frmDialog.GetMemoControl().Lines.Add(s)
                frmDialog.Count = frmDialog.Count + 1
                frmDialog.Show()
            End If
        End Sub
End Class
```

Form this example the code generated by the form designer in the InitializeComponent method was removed.

btnOk_OnClick

In this event the edit box is checked for data. If data exists, it is added to the memo control in the second form and the counter keeping the number of times the second form was shown is increased and the second form is shown.

lnkQuit_OnClick

This event terminates the session.

8.12 Note for C++ Builder users

Due to the way forms work in C++ Builder, the best place to initialize all automatic class instances is inside the form constructor rather than within the OnCreate event handler.

This will avoid the problem of instances being created twice, problem that arises because of the creation order of the form elements in C++ Builder. In C++ Builder the OnCreate event handler is called before the form constructor, and the OnDestroy event handler is called after the form destructor.

To avoid such problems, please move all the automatic class instances initialization in the form constructor, and keep in the OnCreate event handler only the dynamically allocated instances.

Section

9 State Management

9.1 Inherent State

Standard web development tools have automatic session management, but just means that it tracks session info for you. You still have to deal with the state info between pages, or proxy information in and out of a state object. The state objects are also usually limited to strings and data must be marshaled in and out of strings, which is not feasible for complex data types. IntaWeb has something better, and that is inherent state management. What the heck is that you say? Some new buzzword? No. Ask yourself this, How do you manage state in a normal Delphi or Windows application? What? You do not have to? EXACTLY! That is how you manage state in IntaWeb.

9.2 Restrictions

Global Variables

Global variables in general should not be used. If you want to use a global variable that is "global" yet specific to each user session you need use variables that are tied to the user session as described later.
If however you want a variable that is global among all user sessions you can and should in fact use a global variable. However as IntaWeb is a threaded environment you must take the proper steps to protect the variable from concurrent access.

ThreadVars

ThreadVars should never be used in an IntaWeb application except as temporary storage under controlled circumstances. IntaWeb is based on HTTP which is stateless. This essentially means that threads are not assigned to a specific user and a user is moved between threads between HTTP requests.

9.3 Safe Storage

Safe Storage in IntaWeb for Delphi

Form / Datamodule Members

This section refers to IntaWeb for Delphi only: since IntaWeb forms and datamodules are persistent just like in a normal Delphi application you can store information as member variables and properties. Such members should be used when the form itself needs to store data about its instance or to receive input from another form.

User Session

The user session (covered more in detail in the Session Management section of this manual) contains a .Data property that can hold a reference to an object. When you need to store user specific information you can store it in the .Data property of the session. Data accepts a TObject instance and will destroy the TObject automatically when the session is destroyed. The easiest way is to create an object and add the fields that you wish, and then create your

object and store it in the session's Data property when the session is created. The Phonetics demo shows an extended example of this.

When a new IntaWeb project is created a shell user session object is created for you in the ServerController. The default ServerController looks like this:

```
unit ServerController;
{PUBDIST}

interface

uses
  SysUtils, Classes, IWServerControllerBase,
  IWApplication, IWAppForm;

type
  TIWServerController = class(TIWServerControllerBase)
    procedure IWServerControllerBaseNewSession(ASession:
TIWApplication;
      var VMainForm: TIWAppForm);
  private
  public
  end;

  TUserSession = class
  public
  end;

// Procs
  function UserSession: TUserSession;

implementation
{$R *.dfm}

uses
  IWInit;

function UserSession: TUserSession;
begin
  Result := TUserSession(RWebApplication.Data);
end;

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

end.
```

TUserSession is an empty session object that you can add members, properties and methods to. The code to create the TUserSession for each session is also created in the OnNewSession Event.

A function named UserSession also exists for easy access to the object. So if you changed the TUserSession declaration to the following:

```
TUserSession = class
  public
    Username: string;
    Password: string;
  end;
```

You could access these properties elsewhere in your code simply as shown here:

```
UserSession.Username := 'Joe';
LPassword := UserSession.Password;
```

If you do not need a user session you may choose to eliminate it from the code. It is not necessary and is part of the default template simply as a convenience.

The class type of TUserSession can be of any type. For projects that are generated with a datamodule the TUserSession is a descendant of TComponent and not TObject as shown here. TComponent allows the session to own components such as the datamodule and allows for easier cleanup.

Safe Storage in IntaWeb for Visual Studio .NET

User Session

When a new IntaWeb solution is created a shell user session object is created for you in the UserSession file.

UserSession is an empty session object that you can add members, properties and methods to. The code to create the UserSession for each session is also created in the OnNewSession Event of the ServerController.

```
using System;
using Atozed.Intraweb.UserSessionBaseUnit;
using Atozed.Intraweb.Init;

namespace FormData
{
      public class IWUserSession: UserSessionBase
      {
            private System.ComponentModel.Container components = null;

            private void InitializeComponent()
            {
                  components = new System.ComponentModel.Container();
            }

            public IWUserSession()
            {
                  InitializeComponent();
            }

            public static IWUserSession UserSession()
```

```
            {
                  return
(IWUserSession)Atozed.Intraweb.Init.Unit.WebApplication.Data;
            }
      }
}
```

The static method called UserSession exists for easy access to the object. So if you add the following lines to the UserSession:

```
public string Username;
public string Password;
```

You could access these properties elsewhere in your code simply as shown here:

```
IWUserSession.UserSession().Username = 'Joe';
LPassword = IWUserSession.UserSession().Password;
```

If you do not need a user session you may choose to eliminate it from the code. It is not necessary and is part of the default template simply as a convenience.

9.4 Complex State and the Back Button

Many people quickly discover that when building an IntaWeb application the back button in the browser does not work. By default IntaWeb disables the back button and pressing it has no effect. Please note first that this only applies to application mode. In page mode the back button is fully functional. This limitation is because of the way that IntaWeb allows and uses complex state.

Scenario – Normal Application

Imagine a normal application designed to run on the users local computer. It has five different forms, and for some of the forms multiple instances of that form may be created with different data (such as a properties dialog displaying properties about different objects). Imagine now that at any time, without warning or notice to you the programmer, the user can go to any form in the application. But not only can they go to just any form, they can go to any form, in any prior state, even to versions of the forms which have since been removed from memory. After they move to that form, they can even interact with it. How could such an application deal with this?

Here are a few, but certainly not all of the problems:

- Forms may rely on data in databases that no longer exists because the user deleted it.
- Forms may rely on data that has since changed, and the user would be posting old and invalid data.
- Objects that were in memory have changed, or no longer exist.

Back Buttons in non IntaWeb Systems

System not built in IntaWeb usually support back buttons. However it is because they fall into one of these categories:

- **Stateless –** They are completely stateless and reconstruct state between each page. This is usually very inefficient on the server side for weblications and puts considerably extra load on databases because data is read and written unnecessarily.
- **State Streaming** – These types stream the state into and out of each web page. This consumes bandwidth and slows down page accesses. They also cannot use complex data, or usage of complex data causes the same problems described prior.

Even applications that support the back button, such problems are still encountered. However because they allow old data to be posted they must check the data to see if the requested operations can be performed. This adds significantly to the amount of user code except in the simplest of systems. Such systems are typically not weblications, but individual dynamic pages.

IntaWeb is Not Alone

If you try many online banking applications or ordering systems, many of them have the same restrictions, but do not behave as well. Most of them allow you to go back, but will tell you that you have requested expired content. That is certainly very user unfriendly and confusing to non technical people.

Back Button for Historical Purposes

Under limited circumstances the back button can be supported in application mode. It can be enabled for historical purposes. This means that the back button will be enabled, and the user can move backward. However if they try to interact with data on a page that they have reached using the back button they will fail. If the user tries to interact with such a page, a warning will be displayed:

```
You have attempted to post or refresh data from a page that
depends on information that is no longer available to the server
application.

Your attempted changes will be ignored. You will now be
resynchronized to the current place in the application.
```

After this warning is shown, the user will be shown the current form as it was before they used the back button.
This functionality can be turned on by setting the .HistoryEnabled property to true in the server controller.
This warning dialog can also be turned off. To do so set the .ShowResyncWarning property to false in the server controller. If false, instead of seeing the warning dialog the user will simply be resynchronized with the current form.

9.4.1 Using the OnBackButton Event

IntaWeb 5.1 introduces a new event called **OnBackButton**. However, despite its name, it does NOT fire when the back button is pressed in the browser. This is due to the nature of the HTTP protocol and therefore the browser cannot send an event to the server to indicate that this has happened. On the other hand, what can be detected is when a form that has already been sent to the server is re-sent. This is where this event comes into play.

OnBackButton is fired when a form is re-submitted to the server. The event can therefore be used to detect what operation needs to take place if old data is re-sent. Assigning the event effects two properties: **HistoryEnabled** and **ShowResyncWarning**. When an event has been assigned, the first property is automatically set to True whereas the latter is set to False.

IntaWeb works in sequences. Each form that is submitted carries a sequence number with it. When the application is first started, the sequence (or track ID) is set to 0. Every time a form is submitted, this track ID is incremented by 1. Sequences play an important role in back button events.

The signature for the event is:

```
ASubmittedSequence, ACurrentSequence: Integer; AFormName: String; var
VHandled, VExecute: Boolean
```

- **ASubmittedSequence**: Represents the sequence that has been submitted. This will always be LOWER than the **ACurrentSequence** that represents the current sequence.
- **ACurrentSequence**: Represents the current Track ID.
- **AFormName**: Working with sequence numbers, although powerful can become cumbersome. Alternatively, you can use the AFormName parameter to see WHICH form has been re-submitted.
- **VHandled**: When controlling a re-submitted form, you need to set the VHandled property to True, otherwise IntaWeb will understand that the event has not been handled and will display the default action which is to show the Re-sync message.
- **VExecute**: When a form is re-created in the OnBackButton event, you can choose whether you want to execute it or generate it. By default VExecute is FALSE which means that the form will be generated.

For more information and an Delphi example of using the **OnBackButton** take a look at the **BackButton** demo located in the application folder.

Section

10 Session Management

10.1 WebApplication Object

TIWApplication is to an IntaWeb application, what TApplication is to a standard Delphi application. Like the latter, TIWApplication is not a visual component. It does not appear on the property panel and therefore does not have published properties. However, it has a number of public properties and methods that can be accessed via code in the IW application. For each user session, a TIWApplication object is created. It represents the user's "application" or session. In Intraweb for Visual Studio .NET the application class is IWApplication, and has exactly the same role as the TIWApplication has in Intraweb for Delphi.

10.2 Referencing the Session

The users application can be accessed in several ways.

WebApplication Property of the Form

In any event or method of your forms you can simply use WebApplication which will reference the form's WebApplication property. This will meet the requirements in nearly all cases. However some notable exceptions where this property is not accessible are global procedures, TFrames, datamodules, and non IntaWeb classes.

WebApplication Property of a Control

The base IntaWeb control also contains a WebApplication property that can be used when writing custom controls.

WebApplication ThreadVar

In code that is not contained inside of an IntaWeb form or component, the threadvar version can be accessed. This is especially useful in global procedures, or in TFrame code in Delphi Intraweb applications, or user control code in Intraweb applications for Visual Studio .NET.

10.3 Lifetime

A users session is automatically managed by IntaWeb. When a new session is started IntaWeb will create a new instance of a TIWApplication (or IWApplication in Visual Studio .NET) for the user and track it automatically. It can then be used to acquire information about the user, control the users session, or store additional information. No management on the developers part is required to implement session management, or to track the user.

A session exists until it is manually terminated by calling one of TIWApplication's terminate methods, or a timeout occurs. If a user does not access the application within a given time period, the user's session will be destroyed. The default timeout period is 10 minutes, but can be adjusted by changing the SessionTimeout property in the applications ServerController.

Note: To store session related variables, create members of the TUserSession class (UserSession in Intraweb for Visual Studio .ENT).
Note for Delphi users: Do not work with variables declared in the **var** section, as these are global to all threads and won't reflect session data properly.

10.4 Implementation

Sessions are managed automatically by IntaWeb. Sessions are stored in memory on the server and there fore are secure from users who may attempt to modify the session data.

Each session is assigned a unique session ID that is used to identify the session. The session ID is constructed in a secure manner so that session IDs are not predictable and thus prone to hacking. In addition each session is tied to the users browser and if another browser is detected attempting to use the same session an error will be returned.

For further security use the ServerController's RestrictIPs property. This will check the user's IP address against the session and return an error if the IP address changes. This option is false by default and should only be set to true in Intranets or Extranets with controlled clients. This is because some proxy servers such as Microsofts ISA proxy server change IP addresses between HTTP requests for a given user and will cause multiple IP addresses to be seen by the IntaWeb server.

By default the session ID is embedded in each HTML page and tracked with each HTTP request. This allows a single user to have multiple sessions per application. The disadvantage is that once the user is inside the application they cannot leave the application and return to it. Because of this when using this method of session ID tracking any non application web pages must be opened in new windows unless it is in response to the application terminating.

Session tracking can be set to use cookies instead of embedding in the HTML page by setting the ServerController's SessionTrackingMethod property to tmCookie. This will instruct IntaWeb to use cookies to track the user's session instead. The advantage is that the user can move in and out of the application to other web pages with ease. The disadvantage is that many users disable cookies and also that the user can only have one session per application.

10.5 Storing Additional Data

Additional data can be stored in the .Data property and is covered in the State Management section of this manual.

10.6 Session Related Events

The server controller has events related to session management that are fired for session creation and destruction.

OnNewSession

OnNewSession is fired each time a new user session is created. It passes in two arguments, ASession and VMainForm.
ASession is a reference to the newly created session and can be used to query information about the user or modified with custom information such as creating an object to be stored in the .Data property.

VMainForm is passed as a var parameter. It is initialized to nil and if not set the default main form as specified in the project file (dpr) will be used. VMainForm however can be modified based on parameters passed on the start URL, or based on other criteria to specify a main form for the user. To specify an alternate main form simply create it and return its instance in the VMainForm argument. In Visual Studio .NET, VMainForm is initialized to null and passed by reference and it has exactly the same functionality as in Intraweb for Delphi.

OnCloseSession

OnCloseSession is called when a users session is about to be terminated. This occurs either when one of the forms of WebApplication.Terminate is called, or the session has timed out.

10.7 Memory Consumption

The base memory consumption per session is quite low and in most cases is not a major consideration. The actual size can vary from session to session, but the base memory consumption excluding any user data in the .Data property should typically be at maximum 1024 bytes.

Note: This is the typical setting. If you have enough RAM you can safely store as many bytes as you need.

10.8 Component Reference

More information on the methods and properties of the TIWApplication class is available in the IntaWeb help file. The IWApplication class is covered in the Intraweb for Visual Studio .NET help file.

10.9 How does Session Management Work?

When using Application Mode in IntaWeb, session tracking is performed automatically. This allows the user to concentrate on the application and forget about session management. Even though this is all automated, it is good to understand how IntaWeb accomplishes this and see how it can extend it to store certain information.

IntaWeb allows the user to specify how session management is tracked. This provides flexibility since certain situations restrict the user to a certain type of tracking. For example, when the application is deployed to a large corporation, which has certain browser restrictions such as not permitting cookies, another form of session tracking can be used. To provide this flexibility, session tracking can be obtained using one of the following options: Hidden Fields, URL and Cookies.

Developers that are accustomed to implementing session tracking using technologies such as WebBroker are familiar with these since they are the 3 most common (if not the only) way of implementing such a feature. The method used is set as a property in the ServerController (*SessionTracking*).

10.9.1 URL

URL session tracking is the default method. When using this method, the information appears in the URL at all times. When the application is first called, the URL is of the form:

http://xxx.xxx.xxx.xxx:xxxx/{start_command}

After the first call, the appropriate session information will be appended to the URL, which will be:

http://xxx.xxx.xxx.xxx:xxxx/exec_command/Z/YYYYYYYYYYYYYY

There are two important values here to understand, *Z* represents the track ID whereas *YYYYYYYYYYYY* represents the session ID. The track ID represents the "state" in which the application is. When an IW application starts, the track ID is set to 0. Every time a new request is made to the server, the track ID is incremented by 1. Each new request has a higher value than the previous one. This allows IW to know exactly what state it is in.

The session ID is a random value that is generated from unique values so that it is different for each user that accesses the application. This makes it virtually impossible for sessions to be intermixed between two users.

One of the disadvantages of using URL as session tracking is that the user will always see the URL in this format. Any change in the session ID will result in a invalid session message.

10.9.2 Cookies

Cookies is a good alternative when users have cookies enabled in their browser and have no problems using them. Cookies also offer the advantage of not having a long URL and also re-entry options. By allowing re-entry, the application can link to an external website and from that external website link back into the application. For more information regarding this, see the OnReEntry event in the ServerController.

As for the information stored, the same entries (session Id and track ID) as with tmURL as stored in the cookie.

10.9.3 Hidden Fields

Introduced in Intraweb 5.1, hidden fields can now be used to implement session tracking. Each time a new request is made, two additional fields are rendered inside the FORM tag when the page is displayed. Using this method of session tracking, the URL can also be kept "simple" without the need of displaying the track ID and session ID on each request.

What are hidden fields ?

Hidden fields are standard HTML form fields that don't aren't visual. This is, the regular user never sees them.

Hidden fields are rendered to HTML as a <INPUT TYPE="hidden" NAME="my name" VALUE="my value"> tag, where "my name" is the name of the hidden field, and "my value" is the actual value the hidden field is holding.

Hidden fields are contained within HTML forms, yet they don't have a visual appearance. They are very useful for session tracking, as well as for passing data between forms.

Note: When this session tracking method is used, the ExecCmd string will appear on the URL bar if server side resize is active. This behaviour is by design and can't be avoided.

Section

11 Layout Managers and Templates

11.1 What is a Layout Manager?

A layout manager assembles the HTML pieces from each component into a complete HTML page for output to the browser. IntaWeb has a base layout manager that can be descended from to create new layout managers. Currently IntaWeb has two layout managers and in the future, there will be other layout managers to support XML and more. In Intraweb for Delphi the base layout manager is called TIWLayoutMgr and has two descendants: TIWLayoutMgrForm and TIWTemplateProcessorHTML. Intraweb for Visual Studio .NET has the ContainerLayout base layout manager and the two descendants: TemplateProcessorHMTL and LayoutMgrHTML.

11.2 Form Layout

This is the default layout manager. If no layout manager is specified, and implicit layout manager will be created and used. It creates HTML pages that have the same layout and look as the designed form. In Intraweb for Delphi this layout manger is of type TIWLayoutMgrForm and in Intraweb for Visual Studio .NET is of type LayoutMgrForm.

11.3 HTML Templates

HTML Templates

Templates allow for advanced formatting and layout of individual containers (forms, IntaWeb frames or user controls, regions). Templates also allow a web designer to design the layout of containers without using the IDE. In short, templates allow presentation and implementation to be separated. Templates are simply special HTML files.

The use of templates still requires the browser to support HTML and JavaScript.
Any framed controls will be rendered without frames when templates are used. If you wish to have frames in the template, you should frame them by using IFrame or other method in your template.

To use templates create a "Templates" sub directory in your application directory and create a <FormName>.html file. Next, for the form that you wish to apply the template to:

1. Add a HTML template processor on your form. In Intraweb for Delphi this type of processor is called TIWTemplateProcessorHTML component an is placed on the IntaWeb Control tab on the palette. In Intraweb for Visual Studio .NET this processor is TemplateProcessorHTML and is placed on the Intraweb components toolbox item.
2. Set the form's LayoutMgr to the new HTML template processor control.

Most of the template functionality should be self-explanatory by looking at the examples. To see templates in action see the Phonetics Customer Profiler demo.

For each component, the template should contain a tag of the form {%Component.HTMLName%}.
Note: This rule does not apply to non visual controls.

HTMLName in most cases is the same as the name. When the form is generated, the tags will be replaced with the component output. The use of {%%} instead of <> allows for easier editing in WSIWYG HTML editors and is compatible with all HTML editors. The {% %} tag characters are also not considered special characters and therefore are not converted to special tags. By default, a master FORM tag will surround the body to ensure proper function of all input controls and buttons. However, in some cases this can interfere with the HTML. For such cases, see the help topic for TIWTemplateProcessorHTML.MasterFormTag.

There is a special case when building HTML templates for forms which contain regions or IntaWeb frames in Delphi code or user controls in Visual Studio .NET.
For controls on regions, the following situations can occur:
- controls are placed on a region which has it's own subtemplate. In this case, in you do not have to specify tags for the controls on the region, just reference the region thru a tag. At runtime, the controls on this region will be rendered according to the subtemplate. For a Delphi example on subtemplates see the Features demo.
- controls are placed on a region which does not have a subtemplate associated. In this case refer to the controls with:
{%RegionName.ControlHTMLName%}. If you want the region to be rendered as in design time, in the HTML template write only the tag for the region: {%RegionHTMLName%}. Note that in the subtemplates for regions, controls must be referred only with the HTML name: {%Control.HTMLName%}. Please check the ***Summary Note*** at the end of this section for an example on how to reference controls on regions with assigned subtemplates.

Controls placed on IntaWeb frames/user controls can be referenced by:
- {%FrameNameRegionName.ControlHTMLName%} if the parent IntaWeb frame is not rendered thru a subtemplate. RegionName is the name of the region placed on the IntaWeb frame.
- {%FrameName%}: in this case, the IntaWeb frame is rendered according to an assigned subtemplate or, if no subtemplate is assigned, it will have the layout and look established at design time. If you choose this approach, you do not have to reference through tags any of the controls on the IntaWeb frame.

If you wish to use the Borland style tags <#TagName#> instead of the IntaWeb style tags you can set the TagType property to ttBorland. IntaWeb type tags are easier to use with WSYWIG HTML editors.

Templates and Mozilla: It is not recommended to use IWRegions inside <table> structures, due to some issues with positioning in the Mozilla browser. For example this won't be positioned correctly in Mozilla:

```
<html><body>
<table>
<tr>
 <td>test</td>
 <td>{%IWRegion1%}</td>
</tr>
</table>
```

</body></html>

More details on this bug can be found at http://bugzilla.mozilla.org/show_bug.cgi?id=63895 and http://bugzilla.mozilla.org/attachment.cgi?id=76045&action=view.

Summary Note: Controls on regions or frames who have their own templates must be specified individually. Suppose you have a form (IWForm1), and put a region on the form (IWRegion1), an edit box on the form (IWEdit1) and a label on the region (IWLabel1). If you assign templates both to the form and the region, you have to include the controls on the form in the template for the form (IWRegion1 and IWEdit1 in our example), and include the controls on the region in the template for the region (IWLabel1 in our example). An example with regions and subtemplates can be seen in the Features demo.

11.4 System Templates

System templates can be used to modify the look and layout of system messages and dialogs generated by IntaWeb.

System Dialogs

There are two specific template files called IWShowMessage.html and IWException.html. These are used to provide additional formatting to ShowMessage method and for the display of uncaught exceptions. The following tags must be present:

```
{%textMessage%}
{%butnOk%}
```

Note that the template for ShowMessage has no effect when smAlert or smNewWindow is passed to ShowMessage.

System Messages

System Templates support two tags: {%Content%} and {%AppName%} which can be used to display the error message. {%AppName%} is as specified in ServerController.AppName. The tag {%AppID%}. IT refers to the application ID.

IWError.html

System errors are errors that happen outside of your application and in the server portion of IntaWeb. These errors are rare and usually consist of such things as the user entering invalid requests via URLs or trying to access expired sessions. These errors can be handled by creating a template named IWError.html.

Note: This chapter goes only for the Intraweb for Delphi users.

Section

12 Server Controller

12.1 What is the Server Controller?

Every time you create a new IntaWeb Application*, the project wizard creates a form called IWServerController. All IntaWeb Application projects REQUIRE this form.

Although at first it might not seem too important, and when you create you first IW application, you might not even open the form; the ServerController plays a very important role in an IW application. From it, you can control properties such as the port, SSL settings, and much more. This chapter explains the concepts of the ServerController.

** The form is ONLY created when working in APPLICATION mode with IntaWeb for Delphi. When using PageMode, the ModuleController creates a server controller internally.*

12.2 Properties

Most of the relevant properties of the ServerController are published and therefore accessible via the Object Inspector in Delphi or the Properties tab in Visual Studio .NET.

For information on the properties of the ServerController, please see the IntaWeb component reference help file.

Section

13 Writing Custom Components

13.1 Overview

All IntaWeb components are written using an open API that easily allows you to write your own components and add them to IntaWeb just as you can with Delphi or Visual Studio .NET. In Intraweb for Delphi, to further facilitate the writing of components the source code for all components in included, even in the evaluation edition.

13.2 Under Construction

This chapter is still under construction and will be expanded in the future. Please feel free to use our newsgroups to ask any questions related to writing custom components.

13.3 Source Code

Source code is included for most of IntaWeb for Delphi components. The source code can be found in the source subdirectory of the IntaWeb directory. The API is very Delphi like and for most people quite explanatory.

13.4 Core Server

In each IntaWeb application there is a core server which is responsible for serving the actual HTTP requests. This server is of based type TIWServer and is accessible by using the global GIWServer variable in the IWServer unit in the Delphi version or the Atozed.Intraweb.Server namespace in the Visual Studio .NET version. The core server has methods that are of use to component writers, and is documented in the help files as TIWServer.

13.5 Third Party Program

If you are creating components for distribution, either free of charge or for a fee, you should consider joining the IntaWeb Third Party program. Information about this program is available on the IntaWeb website. The IntaWeb Third Party program can help you with the development of your components, as well as the distribution and promotion of your components.

Section

14 Javascript

14.1 Overview

IntaWeb heavily uses Javascript to implement its functionality. However some users who are comfortable with Javascript or want to add custom functionality can add their own Javascript without having to completely bypass IntaWeb. IntaWeb contains many libraries and functions that may be helpful.

This section will not describe every library or function as many are component specific or of little use to the developer. Instead an overview and introduction will be presented for easy integration into the IntaWeb Javascript libraries.

14.2 Areas of Implementation

There are endless places to add Javascript to an IntaWeb application however the common ones are:

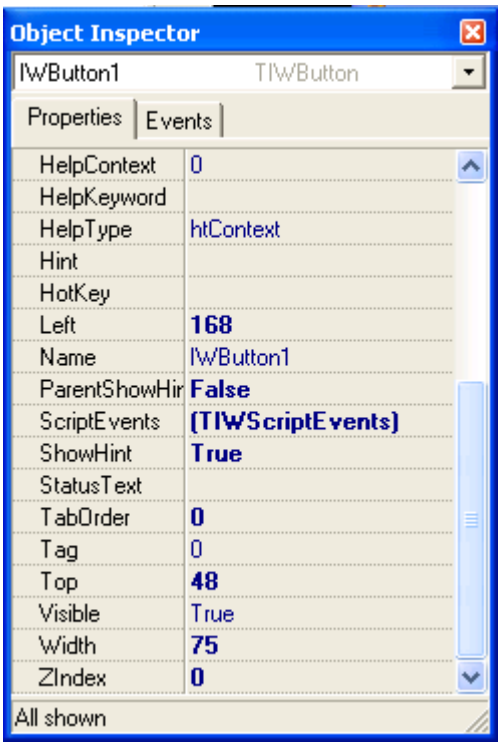
1. Using the forms methods and properties: Javascript, AddToInitProc, AddValidation, ScriptFiles
3. From a custom control.
4. In a template.
5. Using ScriptEvents

14.3 Using ScriptEvents

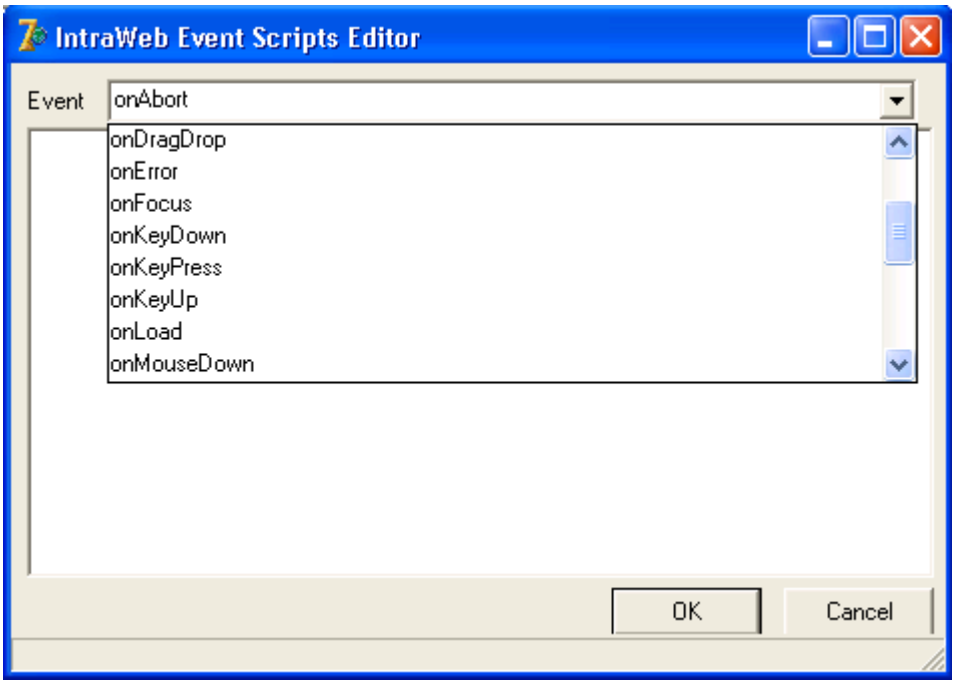
Many IntaWeb visual controls have events associated to them such as the button's OnClick event. This allows you to program the events like a standard application. However, there are some specific JavaScript events that are not available. Although events such as OnBlur, OnFocus, etc are not available directly, you can still use them via the ScriptEvents property of the control.

Accessing ScriptEvents property in Delphi

By double-clicking on this property, the ScriptEvents editor is displayed.

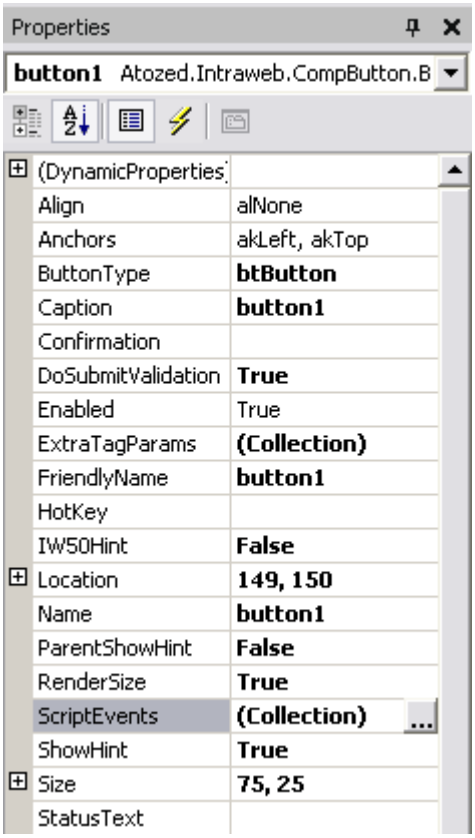


Once loaded, there is a list of events associated to that control. These events correspond to JavaScript events, therefore, any code entered has to be JavaScript code and not Delphi or C++ code.

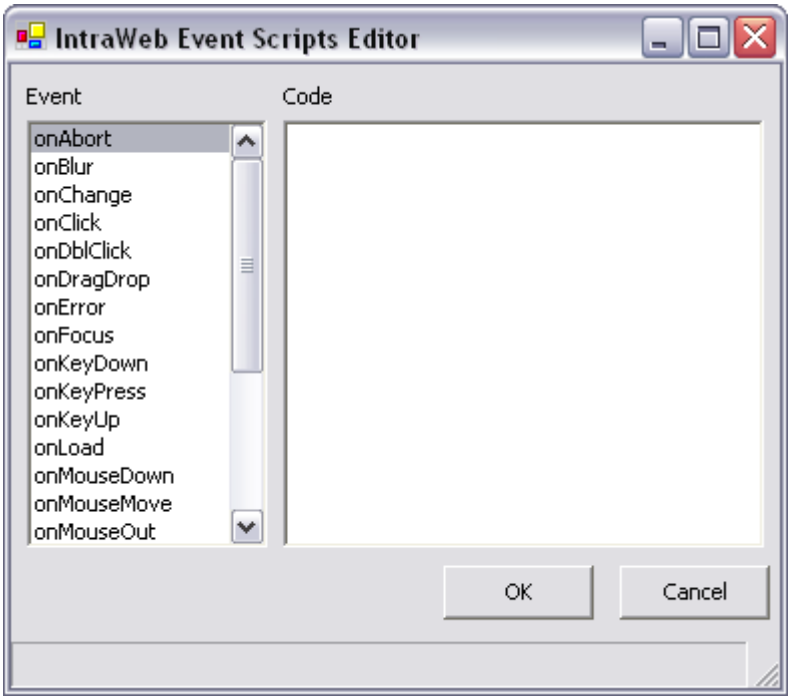


Accessing ScriptEvents property in Visual Studio .NET

Open the ScriptEvents window from the control's properties:



and add the code for the wanted JavaScript event:



To see how to associate an event with one of these, see the next two sections.

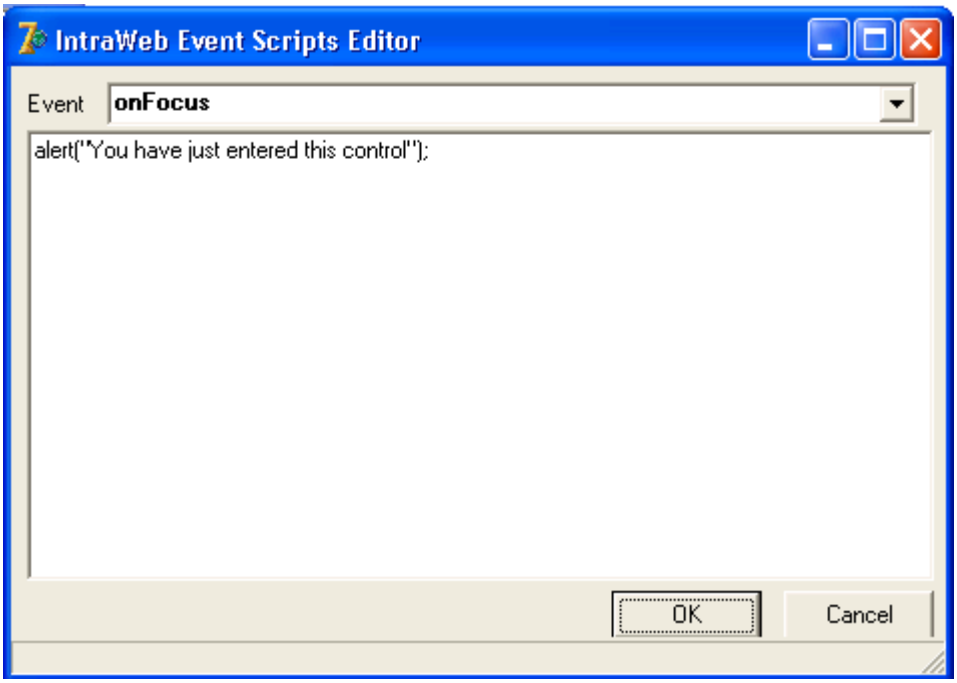
Writing JavaScript events

(Note: Since IW 3.2 controls rely EXCLUSIVELY on HTML and do no use JavaScript or CSS, ScriptEvents is not available for these controls)

14.3.1 Writing JavaScript events

The first step is to chose the event for which you want to associate Javascript code to. In this case, we are going to make an function that displays a simple alert when the control obtains focus.

The first step is to drop a TIWEdit on the form and double-click on the ScriptEvents property. Once the editor is displayed, click on the OnFocus event and then enter the following text in the edit box:



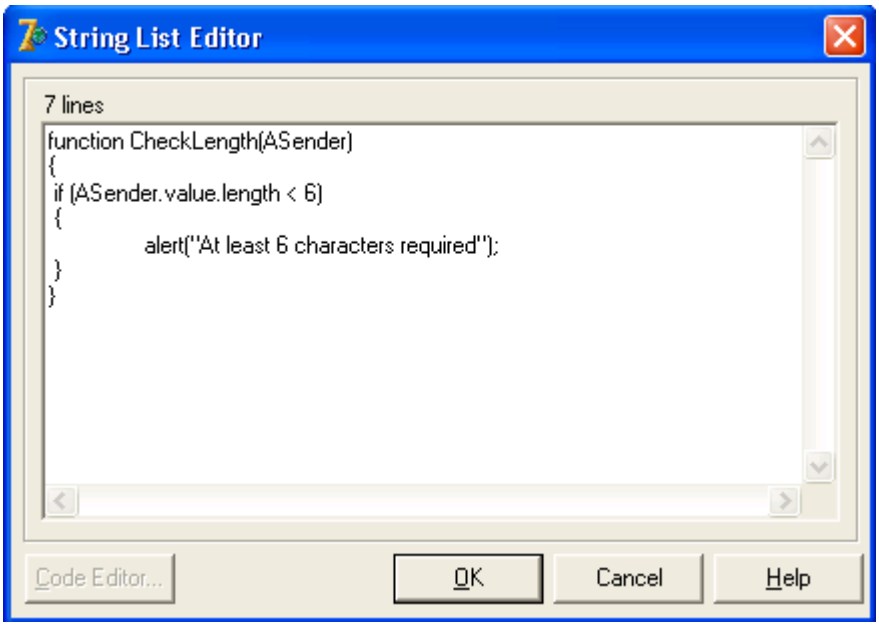
Now when you execute this:

and enter the IWEdit1 box, the alert will show up in the browser.

The next step is to interact with the value entered in the edit box. In this case, we are going to perform a check of the value entered to see if it is at least six characters. The first thing we need to do is define the function that checks to see if the text length is correct:

```
function CheckLength(ASender)
{
 if (ASender.value.length < 6)
 {
   alert("At least 6 characters required");
 }
}
```

Once this function is defined, we need to be able to use it. To do so, we can add it to the JavaScript property of the form:



The last step is to call the function from the OnBlur event of the IWEdit control. To do this, choose OnBlur from the ScriptEvents of the control and enter the following code:

```
CheckLength(this)
```

Where "this" represents the actual control in JavaScript.

Everything that has been done previously, could also be done at runtime using the controls "HookEvents" method. For more information regarding HookEvents, see the source directory.

14.3.2 More Script Events

In the previous section (Writing JavaScript events), we saw how to write events associated with a control and how to interact with the object. In this section, we will look at a couple more examples of writing events.

Let us see an example of how to dynamically change the color of a button when the mouse pointer moves over it. Obviously the events associated to this are OnMouseOver, OnMouseOut. In the OnMouseOver, we place the following code:

```
this.style.color='white';
this.style.backgroundColor='red';
```

As you can see, this time the code has been entered directly into the event as opposed to calling a function (not necessary in this case either). You can also contemplate that more than one instruction can be entered. In the OnMouseOut we place the following code:

```
this.style.color='white';
this.style.backgroundColor='green';
```

Now when you move the mouse over the button, the background will go red and the text white. When the mouse pointer leaves the button, the background color will change to green.

Up to now, we have seen how to associate events with a particular control and how to interact with values/properties of the control. The next step is to see how to interact with other objects on the page.

IW controls are rendered as object named with the following format:

COMPONENT_NAME{IWCL}

For example, if we have a button dropped on the form named IWButton1, the corresponding HTML element would be:

IWBUTTON1IWCL

This way, it is easy to refer to other controls. For example, we can enter the following value in the OnChange event of an IWEdit:

IWBUTTON1IWCL.style.color='brown';

This way, when the value in an IWEdit changes, the button color will automatically change to brown.

14.4 Javascript Functions

14.4.1 Common Functions

IntaWeb includes several script files with JavaScript functions that are used internally. Some of these functions can also be called by the developer to perform certain actions. Following is a list of the more common ones:

FindElem

FindElem takes as parameter an object name and returns the object if it is found. You can use this function to find the instance of a certain element on the form.

SubmitClickConfirm

SubmitClickConfirm is used very often in IntaWeb. It takes fours parameters:

```
function SubmitClickConfirm(AObjectName, AParam, ADoValidation,
AConfirmation)
```

The first is the object name, the second is a parameter passed to the action of the form. The third is whether validation is required or not and the fourth indicates the confirmation string. This function is used every time a confirmation string is entered into any control that supports the property. What happens is the following:

When entering a value in the confirmation property of a TIWButton for example, the following code is called:

```
SubmitClickConfirm("IWBUTTON1", "", True, "Did you really want to click
this button?")
```

What then happens is that SubmitClickConfirm will call a validation box with the text passed as parameter. If the confirmation box returns True, the form will be submitted. Otherwise it will not be submitted.

When adding code prior to a SubmitClickConfirm, make sure that the code you add returns a True or False so that the form is submitted accordingly.

Section

15 Page Mode

Page mode is only available in Delphi 5-7, and C++ Builder. In Delphi 8 and Visual Studio.Net, ASP.net can be used instead of page mode.

15.1 Introduction to Page Mode

IntaWeb can be used in either Application Mode or Page Mode. The latter makes use of other technologies such as WebBroker or WebSnap, allowing the developer to combine existing knowledge with RAD technology. Although for most purposes, application mode is the way to go, Page Mode offers an interesting approach to Web development. To understand the difference between the two, let us see a scenario of where each one would fit in appropriately.

If developing an application (or weblication) where state management is required and each step is interlinked with the previous, such as a contact relational management system, application mode would be most appropriate. This is because the whole application fits nicely into an automata scenario. For example, in normal CRM systems, a user logs in and is presented with a menu from where he/she can access different utilities such as contact management, customer invoices, etc. These are all tied to the same user, where Mr. Smith has certain contacts and Miss Jones has others. Session management plays an important role here and tracking the entire "session" is very important.

On the other hand, if there is a website where there are various independent sections, such as News, Stock quotes and Guest Book, none of these are really linked together. One person might want to see the news whereas another person would like to leave his/her signature. There is no "session tracking" as such, no need to remember if the user that requested news also requested stock quotes. This is where Page Mode fits in perfectly.

The primary difference when working with Page Mode as opposed to Application Mode, is that the former does not offer any kind of state or session management. All this needs to be taken care of by the developer using existing technology such as WebBroker (or WebSnap). Again, in most cases, session management is not required in these cases. It is very important to think through what exactly is the purpose of the web application before deciding on using Page Mode or Application Mode, i.e., is a full-blown application being developed or a dynamic site.

When developing dynamic sites with Page Mode, the technology behind it (as mentioned previously) has to be either WebBroker or WebSnap. IW allows adding to this backbone by providing "pages" to design interfaces. Think of Page Mode as pages where each page can be designed using RAD technology with drag-n-drop of visual IW controls. Therefore, the first step is to decide whether WebBroker or WebSnap is going to be used.

15.2 Working with Page Mode

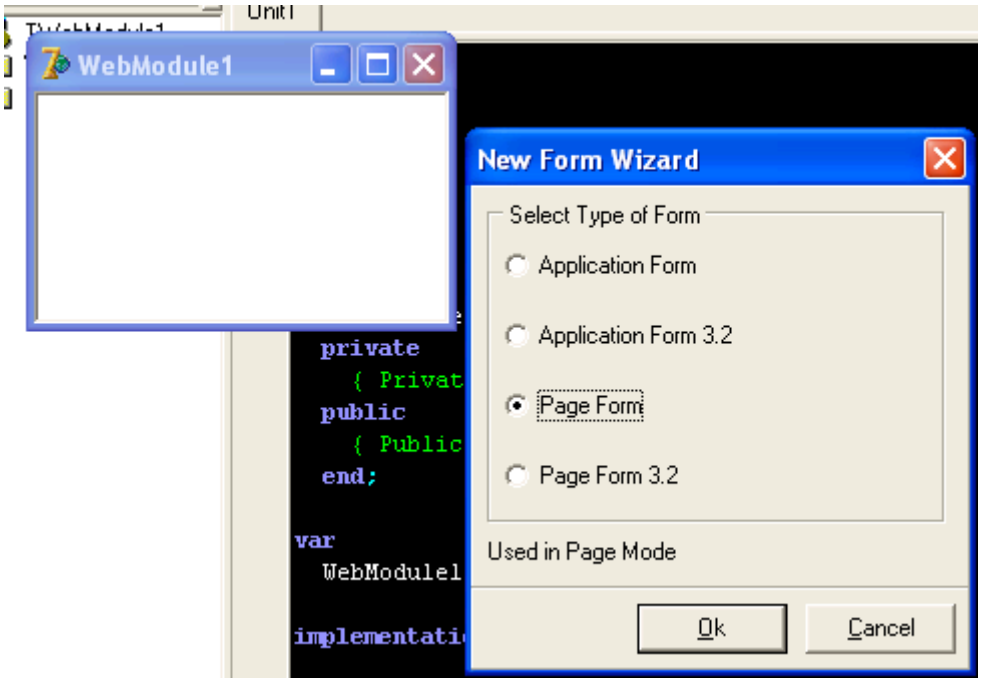
The most important thing to remember when working with PageMode, is that it is basically and add-on to other technologies such as WebBroker or WebSnap. Therefore, it is vital that you know and understand either of these technologies to successfully develop applications using PageMode. On the Atozed website there is a tutorial about WebBroker for beginners.

WebBroker is a low-level interface. The problem is that it is often pitched as a high-level one. PageMode can be considered as "plugins" that sit on top of WebBroker (or WebSnap), which means that you are not actually building an IW application but more a WebBroker application where parts of it are generated using IW. Consider IW in PageMode as a "third-party" add-on to WebBroker.

The first step is to create a new WebBroker application. In this case, an ISAPI DLL will be created. For

more information on WebBroker, see the Delphi online help or tutorials on the Atozed website.

Once the WebBroker application has been created, a new IW form will be added to the project. To do this, use the IntaWeb wizard page in the Object Repository. Depending on whether the application is going to be designed for PDA devices, PageForm 3.2 should be chosen as opposed to PageForm.

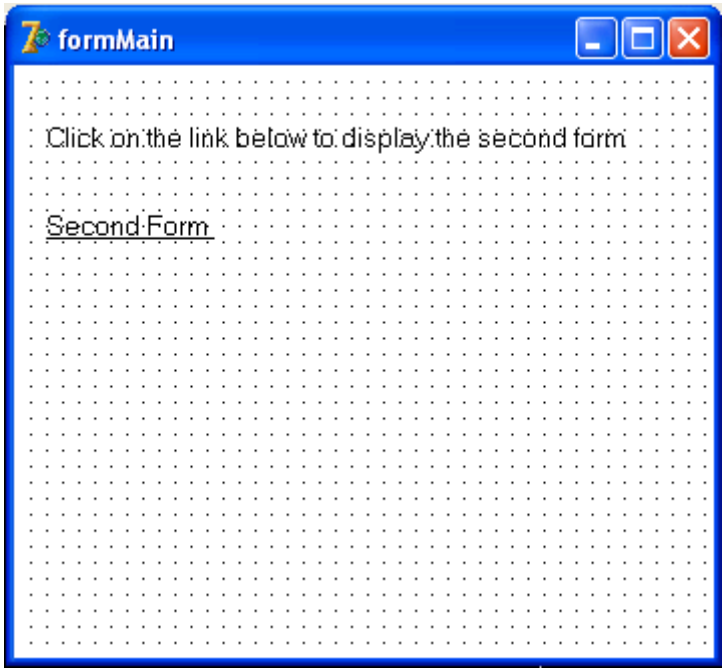


After click on the Ok button a new IW form will be displayed. This form is very similar to the IW forms used in Application Mode, however it has some specific properties that are related to Page Mode. In particular there are three properties that are important:

- AutoProcess
- FormAction
- PostToSelf

What AutoProcess indicates is whether the parameters will be automatically read and matched to components on the form. This in most cases has to be set to True. Since this form represents a actual "HTML form", an important property of any HTML form is the Form Action. This property can be set using the FormAction. However, in most cases, the FormAction would be to call the underlying IWPageForm itself, therefore by setting the PostToSelf property to True, this will be taken care of and FormAction can be left blank. If on the other hand, the post has to be handled by a WebAction defined in the WebModule, PostToSelf would need to be set to False and FormAction would need to contain the appropriate action.

In the case of this example (see PageForms in the Demo directory), PostToSelf is set to True and FormAction is left blank. The next step is to drop some controls on the form.



In this case, a IWLabel and IWURL are placed on the form. The next step is to create a second IW form. Again, using **File -> New -> IntaWeb -> New Form** a new Page Form is created. Before placing any code in either form, some previous steps are required.

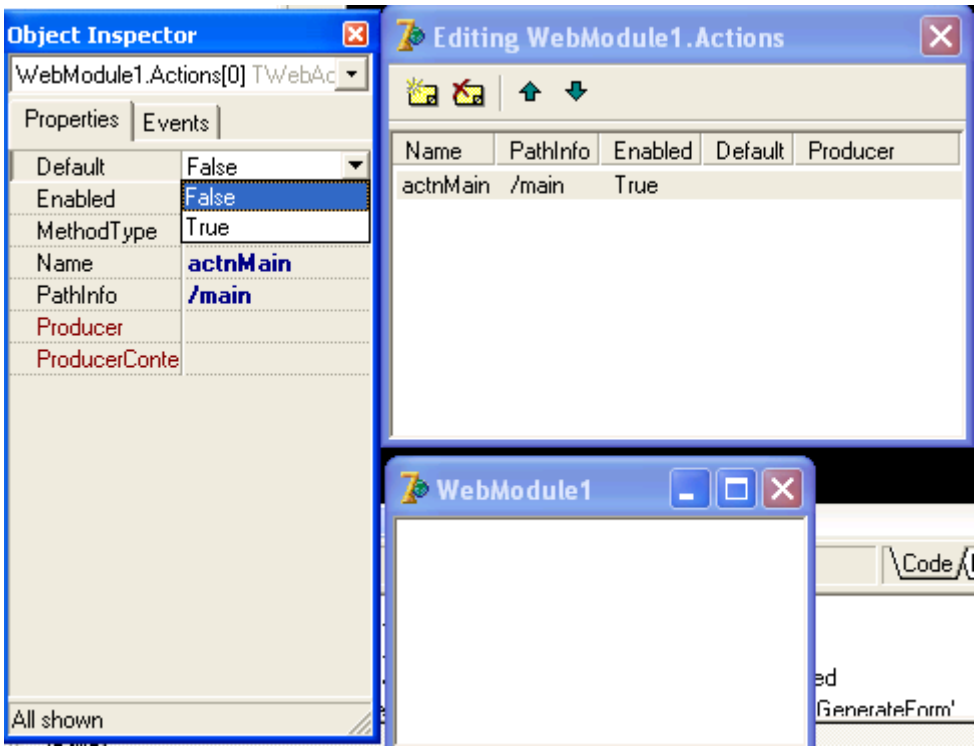
Note: *Some units might not be included automatically in the new page form. If the compiler complaints about undeclared symbols, check if they aren't in this units first: IWApplication, IWTemplateProcessorHTML, IWLayoutMgr and/or IWHTMLControls.*

When using PageMode, the underlying technology takes care of displaying the forms. In Application Mode a form would be displayed using something like:

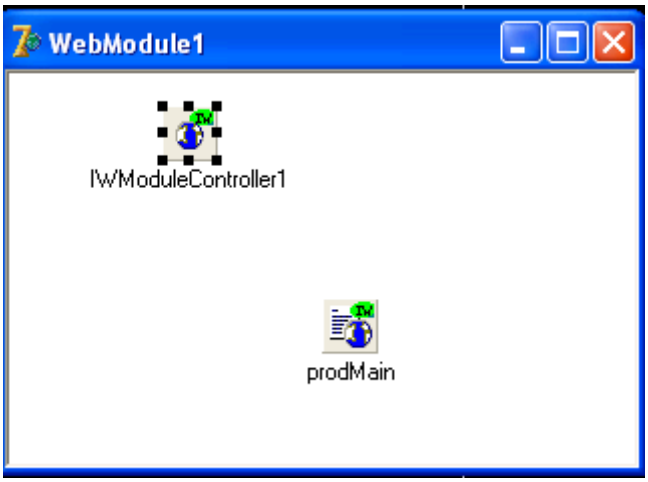
TIWForm.Create(WebApplication).Show;

In the case of PageMode this is a slightly different.

To display the first form, define a WebAction in the WebModule as /main:



Set the Default value to True and define the PathInfo and Name (as displayed in the figure above). As with any normal WebBroker application, there are two ways to send a response back. One is using the property *Producer* and the other is by setting it in code. In this case, the same thing can be done to display the IW form. However, there are two previous steps required. The first is one-off per application and that is to use a IWModuleController. This is a component that automatically creates an IWServerController. All that needs to be done is to drop ONE of these components on the WebModule:



There are no additional properties or events that need to be assigned. Just placing one on the WebModule is sufficient.

The next step required to display an IW form is to use a IWPageProducer component. ONE is required PER form (or assign it dynamically at runtime). This component has only one event which is the *OnGetForm*. This event returns the contents of the IW form to the broker:

```
procedure TWebModule1.prodMainGetForm(ASender: TIWPageProducer;
  AWebApplication: TIWApplication; var VForm: TIWPageForm);
begin
  VForm := TformMain.Create(AWebApplication);
end;
```

This is similar to what is used in Application Mode, except that here, instead of doing a Show, the result of creating the form is assigned to the VForm parameter. The next step is to assign the producer to the WebAction using the Object Inspector. Since there is a second form in this application, the same steps are performed for the second form. As can be seen from the event, the owner of the form is also passed in as a parameter (AWebApplication). A second WebAction needs to be defined to display the second form. This is again done in a similar manner to the first WebAction.

Note: When adding a IWPageProducer and assigning the event in the WebModule, two units have to be added to the uses clause, IWApplication and IWPageForm (IWPageForm32).

What is left is to display the second form from the first one using an IWURL. To do this, all that is needed is to call the WebAction that produces the second form. The URL can be assigned to the IWURL in the form's OnCreate event:

```
procedure TformMain.IWPageFormCreate(Sender: TObject);
begin
  IWURL1.URL := WebApplication.URLBase + '/second';
end;
```

The second form prompts for a name and displays a label. This is programmed exactly the same as in application mode.

Therefore you can see that working with PageMode gives you the RAD flexibility of IW and allows you to work with existing technologies. Much of the way things are done coincides with Application Mode, apart from some minor differences that have been mentioned in this example.

15.3 IntraWeb and Websnap

This sections provides a brief introduction to how IntaWeb can be integrated with WebSnap. A demo will be built that uses WebSnap to provide the framework, login, and session management. IntaWeb will be used to provide the user interface. In this manner of integration the products are quite complimentary.

The demo is a simple demo that takes a survey of two questions that are of vital importance to the programming community. The two questions are:

1. Which was the BEST Star Trek movie?
2. Which was the WORST Star Trek movie?

It will then collect your vote and tabulate it with other voters. To see this, simply run the demo in the browser multiple times. After it tabulates the votes it will generate a small chart displaying the results. Full source code for the demo is included in the demos directory. We have designed it to be simple as possible so as to make it easy to follow. It demonstrates the following:

1. IntaWeb integration with WebSnap.
2. Use of IntaWeb Page Mode.
3. Use of WebSnap session management with IntaWeb.

4. Use of WebSnap for control of authentication.
5. Use of IntaWeb to provide the primary web interface.

15.3.1 Creating the Demo

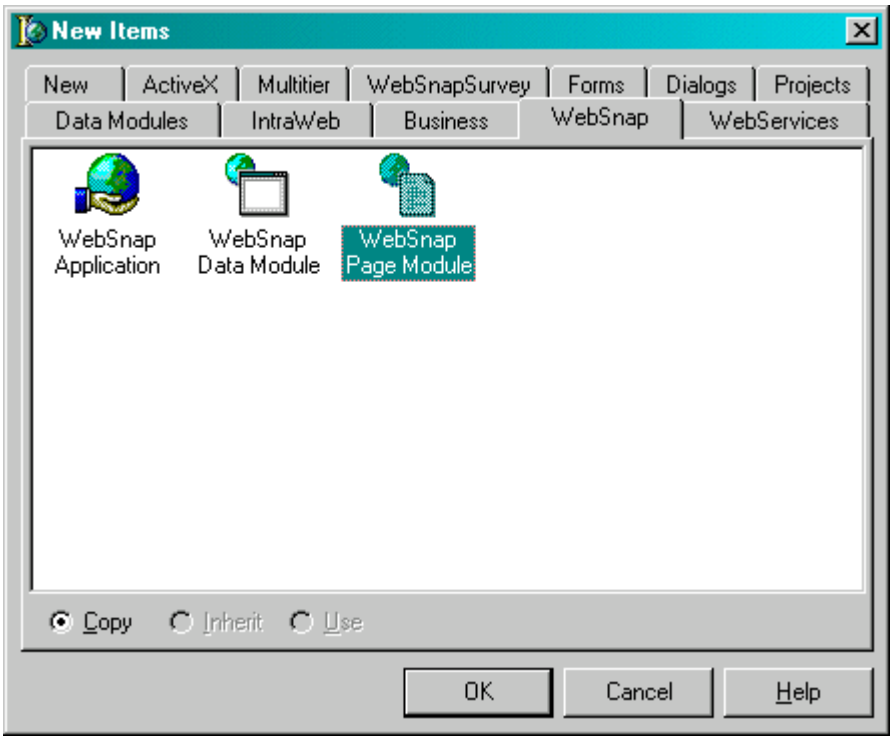
It is assumed that you are familiar with WebSnap and thus we will just show the IntaWeb specific parts in creating this demo. We started with a standard WebSnap application that contained login support using a TWebUserList and a TLoginFormAdapter.

The first thing that must be done to use IntaWeb with WebSnap is to add a TIWModuleController. To simplify distribution and not require distribution of external files, IntaWeb serves "internal" files from its libraries. IntaWeb has several internal files and as a user you can add more using IntraWeb's API.

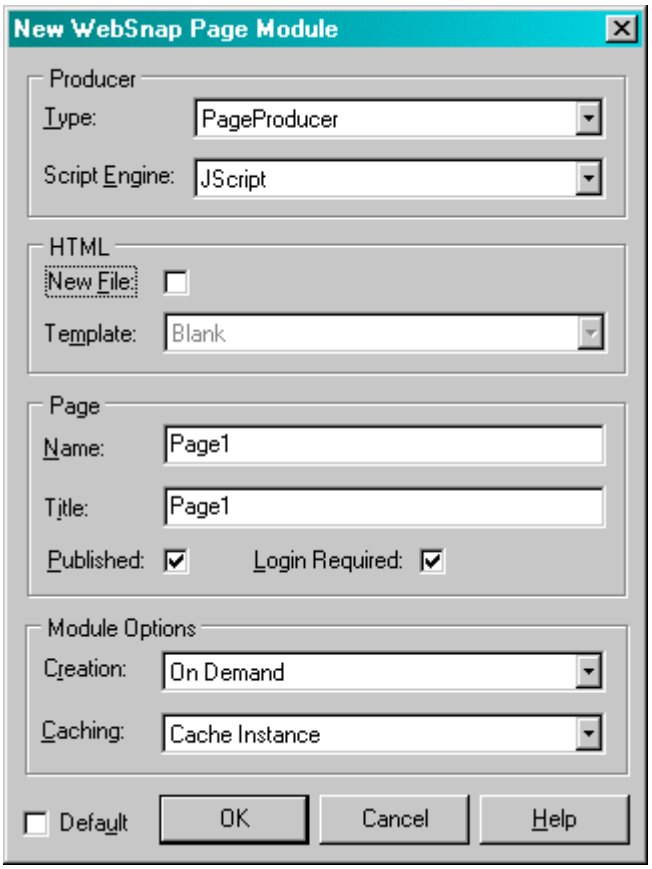
TIWModuleController hooks into WebSnap's dispatch mechanism and provides this functionality and other core IntaWeb requirements. This component can also be used to use IntaWeb with WebBroker and is demonstrated in the GuessWB demo that is provided with IntaWeb.

For the TIWModuleController to be effective, the application module also needs a TWebDispatcher. If you application doesn't already have a TWebDispatcher, simply open the application module and add a TWebDispatcher (from the Internet tab) and then a TIWModuleController (from the IntaWeb Control tab). No further changes are required, IntaWeb and WebSnap will do the rest.
Your application module should now look like this:

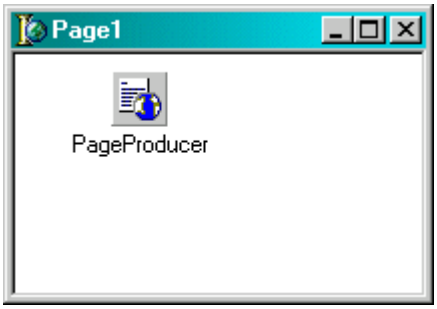
Next we created a new WebSnap page module. To do this we selected *File : New : Other : WebSnap tab : WebSnap Page Module*. The dialog is shown here:



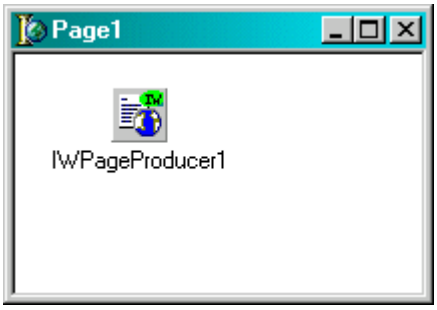
After OK is clicked, Delphi will display the New WebSnap Page Module dialog as shown here:



Make the settings match the settings as shown in the figure above and select OK. Delphi will now create a new WebSnap Page Module. It should look like this:



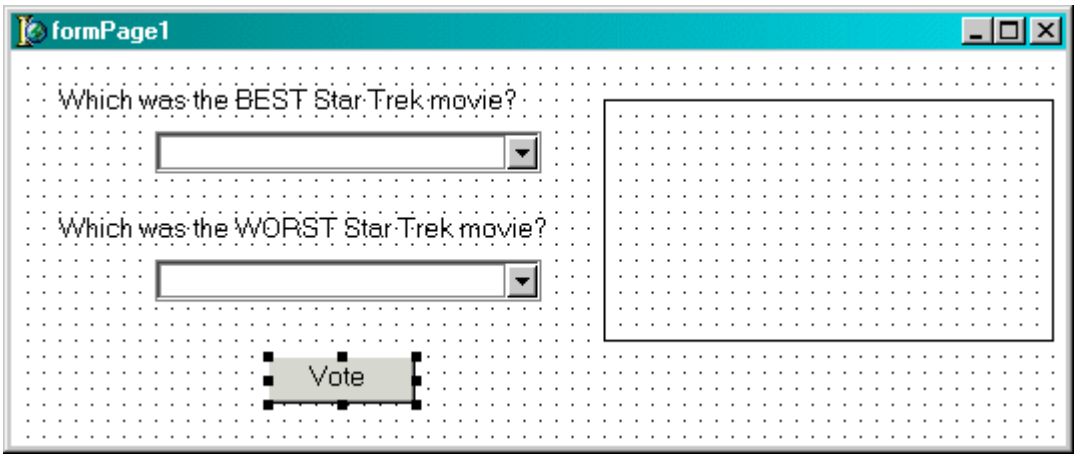
Delete the TPageProducer and create a TIWPageProducer (from the IntaWeb Control tab). The page module should now look like this:



Save the page module and name it Page1Module.pas. Now we need to create an IntaWeb page form. Select *File : New : Other : IntaWeb : New Form*. Select the Page Form radio button then click OK. Save the form as Page1Form.pas. Now lets go back and link Page1Module to Page1Form. To do this create an OnGetForm event for the TIWPageProducer. The event needs to look like this:

```
procedure TPage1.IWPageProducer1GetForm(ASender: TIWPageProducer;
  AWebApplication: TIWApplication; var VForm: TIWPageForm);
begin
  VForm := TformPage1.Create(AWebApplication);
end;
```

This creates an instance of TformPage1 on demand. So that the unit will compile IWApplication and IWPageForm must also be added to the uses clause.
Now let's go back to Page1Form and create our survey questions. We've created two TIWLabel components, two TIWComboboxes, one TIWButton, and one TIWText. For the combo boxes we have also set the RequireInput = False. Our Page1Form now looks like this:



Next we will add the code for the form's OnCreate event. Double click on the form and enter this code. The code merely loads the text and identifying numbers into the combo boxes.

```
procedure TformPage1.IWPageFormCreate(Sender: TObject);
var
  i: TSTMovie;
begin
  for i := Low(i) to High(i) do begin
    cmboBest.Items.AddObject(GMovies[i], TObject(i));
  end;
  (cmboBest.Items);
end;
```

Now we will add an OnClick event for the button. Double click on the button and add this code:

```
procedure TformPage1.butnVoteClick(Sender: Tobject);
var
  LBest: TSTMovie;
  LWorst: TSTMovie;
begin
  LBest := miMotionPicture;
  LWorst := miMotionPicture;
  if cmboBest.ItemIndex = -1 then begin
    textMsg.Lines.Text :=
      'Please select a choice for best Star Trek movie.';
  end else if cmboWorst.ItemIndex = -1 then begin
    textMsg.Lines.Text :=
      'Please select a choice for worst Star Trek movie.';
  end else begin
    LBest := TSTMovie([cmboBest.ItemIndex]);
    LWorst := TSTMovie([cmboWorst.ItemIndex]);
    if LBest = LWorst then begin
      textMsg.Lines.Text := 'Sorry - but you cannot pick the same movie for
                            best and worst.';
    end else begin
      if WebContext.Session.Values['Confirm'] <> 'Y' then begin
        if LBest = miFinalFrontier then begin
          textMsg.Lines.Text := 'Ugh. The Final Frontier was truly horrid.
                                 Are you sure that is'
          + ' your choice for best?';
```

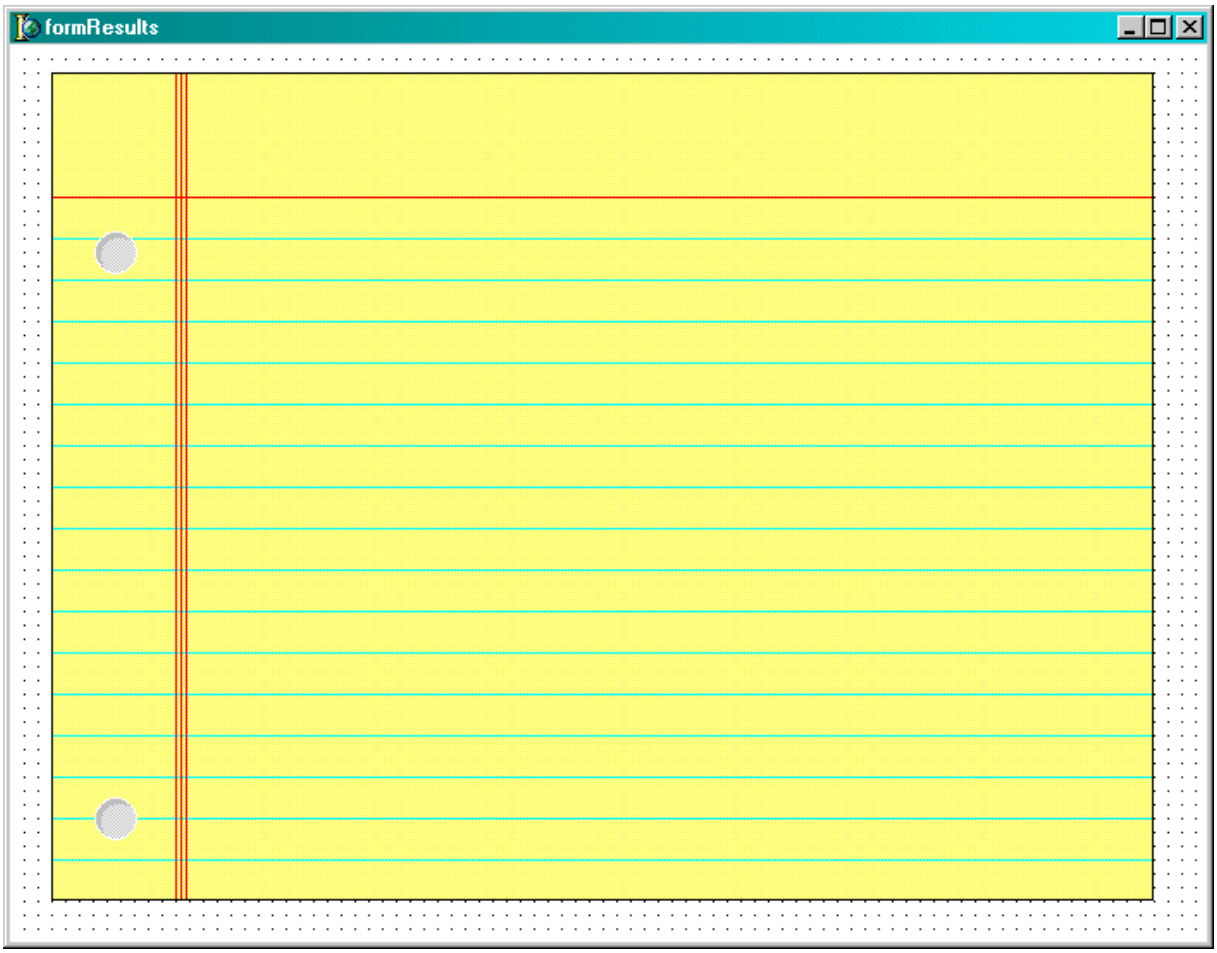
```
    butnVote.Caption := 'Vote with my questionable choice anyways';
            WebContext.Session.Values['Confirm'] := 'Y';
          end else if LBest = miVoyageHome then begin
            textMsg.Lines.Text := 'Good choice! The Voyage home was good
                                  wasn''t it?';
            butnVote.Caption := 'Record my vote!';
            WebContext.Session.Values['Confirm'] := 'Y';
          end;
        end;
      end;
    end;
    textMsg.Visible := textMsg.Lines.Count > 0;
    if not textMsg.Visible then begin
      RecordVote(LBest, Lworst);
      ProduceResponse := False;
      DispatchPageName('PageResults', WebContext.Response, []);
    end;
  end;
```

Now we could spend a lot of time explaining the above code. But did you notice something? Its all standard Delphi code! So we'll just explain a few lines of interest.
The code checks to see if the user has selected information, and also makes sure that they do not select the same movie for both choices. It also enters in its personal opinion about certain choices and displays messages to the user by making the TIWText component visible. If the TIWText component is not made visible, not messages are displayed and all is well. In this case the code calls RecordVote which is a procedure in Global.pas which is part of the demo. It then sets ProduceResponse to False. This tells IntaWeb not to render this page because we will render it manually, or give WebSnap instructions to do so. Finally we give WebSnap instructions to render a different page module to display the results.

There are a few properties on the form itself that we must set as well.

1. Set PostToSelf to true. This instructs the form to generate links that will send the data back to this same form. FormAction can be set if you wish the data to be submitted to another form. FormAction and PostToSelf (When true) are mutually exclusive.
2. Set AutoProcess to true. This instructs the form to automatically parse the HTTP variables and set the component states accordingly. If you wish to control this process manually, you would leave AutoProcess to false.

Next we will create another Page Module and Page Form. The steps are pretty much like the previous one so we will not waste space on this. Instead we will start with a bank page form, PageResultsForm.pas. We have added one TIWImage and loaded a bitmap into it. It looks like this:



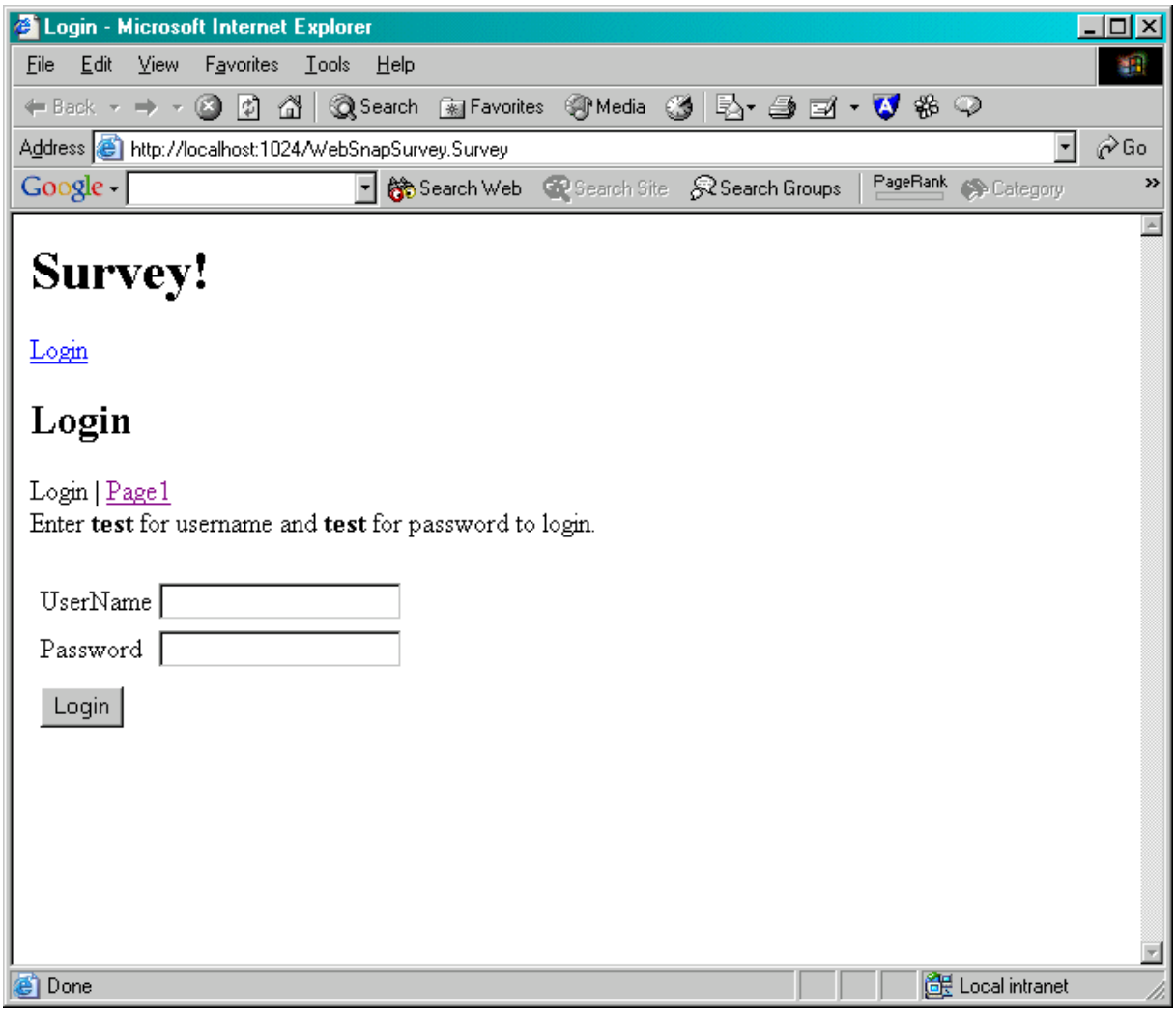
For this form we have created only one event. We have put some drawing code in the OnRender event. The OnRender event occurs each time IntaWeb renders a form, prior to it actually being rendered. Here is the code for the OnRender:

```
procedure TformResults.IWPageFormRender(Sender: Tobject);
var
  i: TSTMovie;
  LMaxBest: Integer;
  LMaxWorst: Integer;
  LMaxWidth: Integer;
  LVotesBest: Tlist;
  LVotesWorst: Tlist;
begin
  LMaxBest := 0;
  LMaxWorst := 0;
  LMaxWidth := 0;
  LVotesBest := TList.Create; try
    LVotesWorst := TList.Create; try
      GetVotes(LVotesBest, LvotesWorst);
      with imagResults.Picture.Bitmap.Canvas do begin
        Brush.Style := bsClear;
        Font.Color := clBlue;
        Font.Name := 'Script';
```

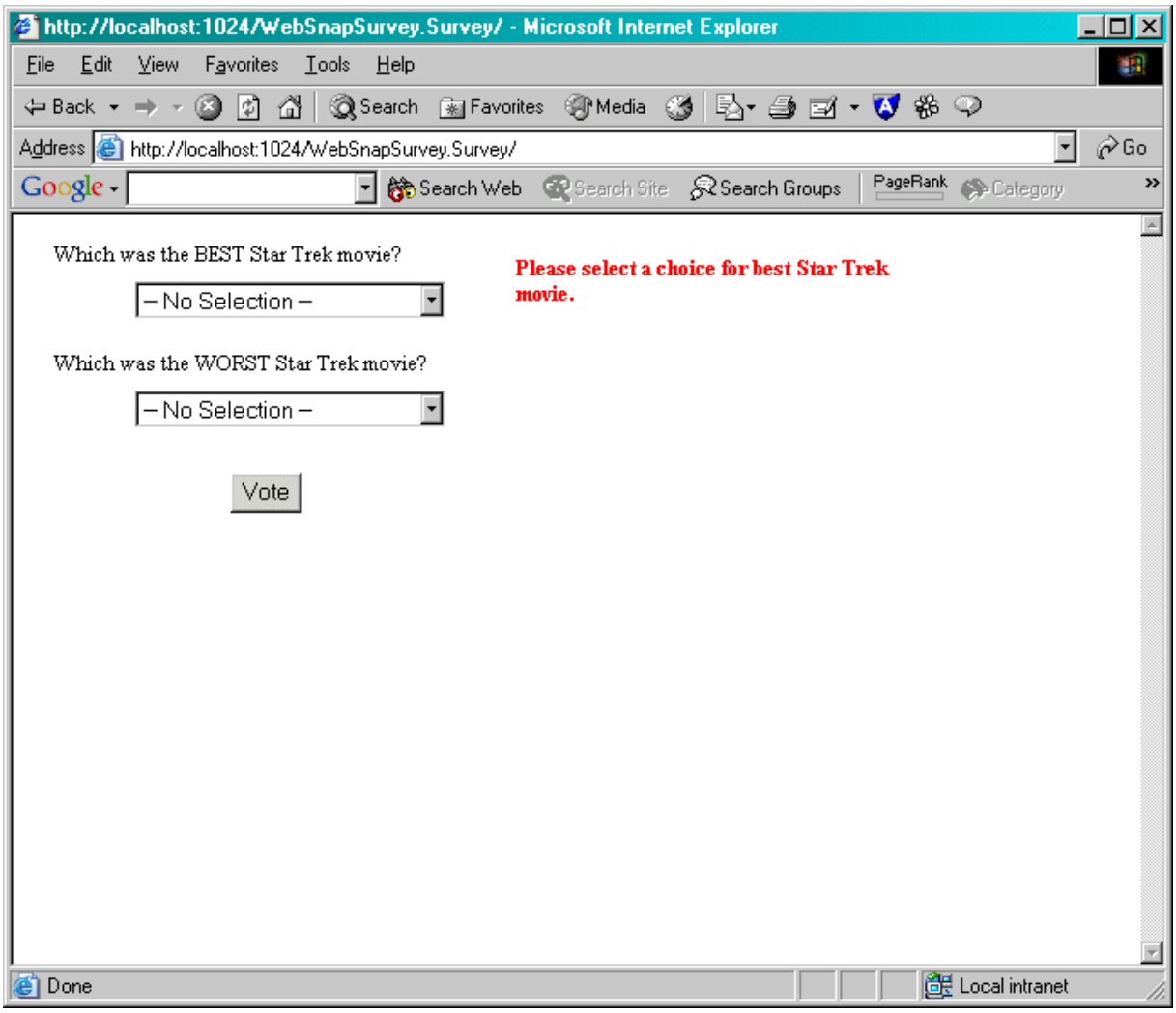
```
Font.Size := 18;
      for i := Low(i) to High(i) do begin
        TextOut(85, 98 + 24 * Ord(i), Gmovies[i]);
        LMaxWidth := Max(LMaxWidth, TextWidth(GMovies[i]));
        LMaxBest := Max(LMaxBest, Integer(LVotesBest[Ord(i)]));
        LMaxWorst := Max(LMaxWorst, Integer(LVotesWorst[Ord(i)]));
      end;
      TextOut(330, 74, 'Best');
      TextOut(480, 74, 'Worst');
      //
      Brush.Style := bsSolid;
      for i := Low(i) to High(i) do begin
        Brush.Color := Gcolors[i];
        FillRect(Rect(310, 98 + 24 * Ord(i)
         , 310 + Trunc((Integer(LVotesBest[Ord(i)]) / LMaxBest) * 150)
         , 98 + 24 * Ord(i) + 20));
        Brush.Color := GColors[TSTMovie(Ord(High(i)) – Ord(i))];
        FillRect(Rect(480, 98 + 24 * Ord(i)
         , 480 + Trunc((Integer(LVotesWorst[Ord(i)]) / LMaxWorst) * 150)
         , 98 + 24 * Ord(i) + 20));
      end;
    end;
  finally FreeAndNil(LVotesWorst); end;
 finally FreeAndNil(LVotesBest); end;
end;
```

15.3.2 Running the Demo

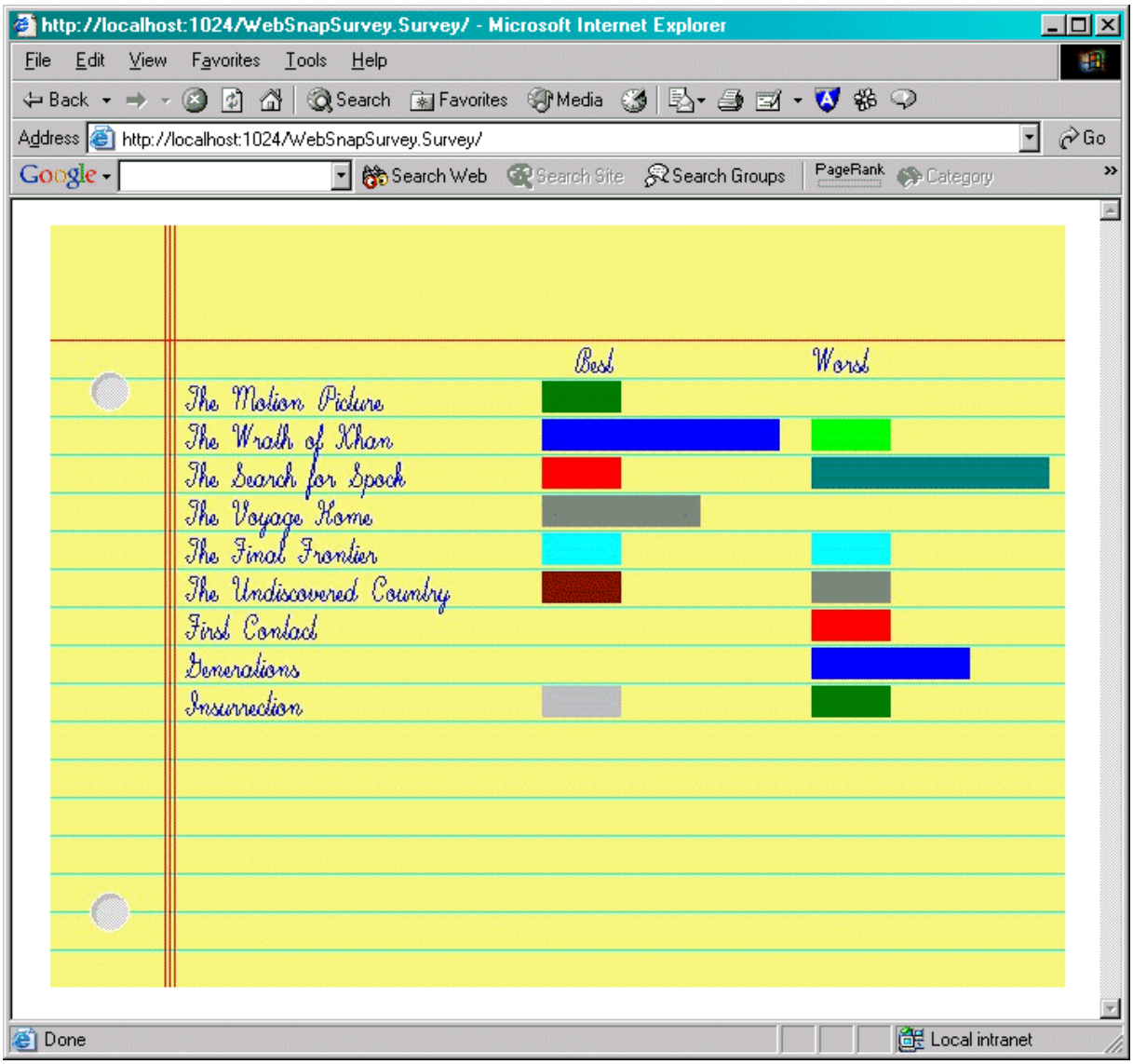
We have now covered the important parts of the demo itself. Let's see what it looks like when we run it. First compile and run the demo and then run the Web Application Debugger from the Tools menu. From the Web Application Debugger click on the URL link and then select WebSnapSurvey.Survey in the browser. This will start our demo application. It should look like this:



This screen is produced by WebSnaps login adapter. Enter test for the user name and test for the password and click Login. This screen will now appear:



Now select your choices and click vote. Now it will display the result screen:



15.4 IntraWeb Pages (IWP)

15.4.1 Introduction to IWP

IntaWeb version 6 introduces IWP's (IntaWeb Pages). You can think of IWP's as server-side pages that are processed automatically by an ISAPI/DSO application. They are similar to the same philosophy as *asp* pages.

The concept behind IWP is that you can create both static and dynamic HTML pages with any extension you want (by default IWP is used although you do not need to specify this anywhere in your application) and the web-server processes the pages and returns either dynamic or static content depending on the nature of the page. Think of it as a reverse process to what PageMode is. To expand a bit more on this, let us first see how PageMode works.

PageMode
The first thing you need to do when developing PageMode applications is to create your

WebBroker/WebSnap application and add IWPageForm (alternatively for 3.2 you would use IWPageForm32) forms to your project. You would then design the form and add the dynamic functionality to these. Once complete, you would use a TIWPageProducer to return the content to the client (browser). Optionally you could use a template to customize the form.

IWP works on the same principle, except in this case, your first step is to create your HTML template and then your form. Although you could do it in reverse order, it makes more sense this way since IWP's *pull* the content from the template. That is, in normal template processing, the IWForm descendant looks for a template and replaces the controls with those on the form. In IWP's, the TEMPLATE looks for the controls and they are replaced at runtime. The template is pulling the information from the form.

IWP is not only limited to the previous, but it also makes the task of developing dynamic-driven websites even more simple than it is now using PageMode. Let us see a step-by-step approach on how to make a IWP website.

15.4.2 Creating the Project

IWP pages work with any type of Delphi (CBuilder or Kylix) internet framework you want to use (like all PageMode applications). You can use WebSnap or WebBroker for example.

The first step is to create a new Web Server application. Once you have the WebModule, you need to drop 2 components on the form. The first one, like all PageMode applications with IW is the *IWModuleController*. The second one is either a *TIWPageController* or a *TIWPageController32* depending on whether you are creating a 4.0 or a 3.2 project. Obviously you could combine both types of applications in one project by dropping one of each component.

The *TIWPageController* is a simple yet powerful component. There are no events and only 2 important properties. As we will see later on, this component handles are the necessary tasks to find the corresponding IWP page, parse the contents, generate the form and return the result to the browser. There is no need for a *TIWTemplateProcessor* or a *TIWPageProducer*. These are both inherent to the component and are transparently used. The only required properties are reflected in the figure below:

The *TagType* property indicates what type of tag you will be using in your IWP pages. By default this is the *ttIntraWeb* tag (which is represented by enclosing control names in {%....%}). You can also set this to Borland style tags. The other relevant property is the *ApplicationURL* which specifies the virtual URL path to your application. So for example, if your script directory in your web server is */scripts* and your project is called *project1.dll*, then this property would take the value */scripts/project1.dll/* (note the prefixing and suffixing of the / character).

That is all you need to do with these properties. The next step is to save the project and optionally set the output path to that of your physical *scripts* directory location.

Once this is done, we can move on to configuring the server to work with IWP and then continue with the design of our pages.

15.4.3 Preparing the server

One of the steps required when working with IWP's is to configure your web-server to identify these files as a "special" file that requires server-side processing.

Below are the instructions on how to accomplish this with both Omni HTTP and IIS.

Configuring Internet Information Server

Open up the Internet Information Services Manager and choose the website you want to deploy IWP's to. Right-click and choose *Properties*.

Click on the *Home Directory* tab and click on the *Configuration* button. If this is disabled, you need to first create an application area. Once the configuration dialog is open, choose the *Mappings* tab and click on the *Add* button. This will present the dialog to add the IWP application mapping.

In the Executable box, we need to enter the path to our project. In our case we can enter *c:\inetpub\scripts\project1.dll*. The next step is to define the extension we are going to use. By default you can use *.iwp*, however, this can be anything you want. Once completed, click OK on all boxes.

The next optional step is to add the index to the default document properties. This is so that if you place the file in a folder with the name *index.iwp* for example, IIS will identify this as a valid request and return the page to you IF the URL does NOT contain the name of the file (for example, http://www.mydomain.com).

Important: You will not be able to validate the entry unless the executable exists.

Configuring Omni HTTPD

Configuring OmniHTTPD is basically the same as IIS. The first step is to add the project application that will be handling IWP pages. Launch Omni and click on *Properties*. Next click on *Web Server Global Settings*.

Click on the *External* tab and enter the extension in the *Virtual* edit box. In the *Actual* box, enter the location of the project that is going to handle the IWP pages. Once complete, click on the *Add* button and next click on *OK*. The next step is to add the IWP entry to the *MIME* settings. To do so, click on the *MIME* tab and like before enter the settings as below:

Click *Add* and then *OK*. That is it. Omni is now configured to handle IWP pages. Remember that if you choose a different extension, you must enter a different value in the corresponding extension boxes.

15.4.4 Creating an IWP page

IWP pages are like normal HTML pages, in fact they are normal HTML pages. The only difference is that optionally they include a special header section and tags for representing controls. This is optional since as mentioned previously, IWP can be dynamic or static. In the latter case, there is no need to add anything other than your normal HTML code.

Here is an example of the HTML source for a static IWP page:

```
<html>
<body>
This is my first IWP static page
</body>
</html>
```

As we can see, there is nothing special about the previous page. This page is returned as is by the IWP application since it is identified as static.

However, the true power of IWP pages is when they return dynamic content that interacts with the server-side PageMode application. When providing dynamic content, you need to add a special header section to the IWP page. This section is enclosed in *<intraweb:header>* tags and currently has two parameters: *FormName* and *Handler*. An example of this would be:

```
<intraweb:header>
FormName=IWForm1
Handler=OtherPage
</intraweb:header>
```

The *FormName* and *Handler* parameter are not both required. Before explaining what each one does, let us see how IWP pages work. The *IWPageController* that was dropped on the *WebModule* when creating the project is the engine behind IWP. It takes care of registering all forms in the project, searching for the correct form when a request comes in and returning the contents, among many other tasks. However, all this is not done by ways of magic! What this component does need to know for each request that is sent to the server is what the corresponding form name or handler is for that page.

If the *FormName* parameter is specified in the header section, the *IWPageController* will search the project for the corresponding *IWPageForm* with the same name and use that for subsequent processing. If the *FormName* is not specified, it will then look for a *Handler*. What is a *Handler*? It is a *WebAction*.

Defining WebActions
Although not necessary, IWP allows you to define a webaction to handle a form (page). Like normal PageMode applications, you can define a WebAction that uses a *TIWPageProducer* to return the contents of the form and perform at the same time other additional actions. However, this is not required and in a way adds an overhead (as far as development time is concerned) when using IWP). You can do it if you specifically require, otherwise, by just specifying the *FormName* in the header section, you can let IWP take care of all the processing for you.

Therefore, for every dynamic page content you want to add to your website, that has a corresponding *IWPageForm (32)*, add the *<intraweb:header>* section along with the *FormName* parameter. This tag is transparent to HTML and when the page has been processed and returned to the browser, this section is stripped out.

Creating a dynamic IWP page
The page below is a simple example of an IWP dynamic page, yet is ideal for demonstration

purposes. Basically it consists of a static web page with a dynamic label that returns the current date and time. You can use any web page development tool to create this.

```
<html>
<head>
<intraweb:section>
FormName=IWForm1
</intraweb:section>
</head>
<body>
This is my first dynamic IWP page. If you look below, you will see the
current
date and time.
<BR>
<BR>
Current Date and Time: {%IWLabel1%}

Bye!
</body>
</html>
```

The next step is to create the corresponding *IWPageForm* for this in our project

15.4.5 Creating an IWP form

As mentioned previously, IWP websites can have both static and dynamic content. Forms are ONLY required when the IWP page returns dynamic content.

In Creating an IWP page we saw how to create the HTML page for our dynamic IWP page. Here we are going to see how to provide the dynamic content.

The first step is to add a new *IWPageForm* to our Web Server project. Again, if designing 3.2 applications, you would need to create a *IWPageForm32* form.

Create a new *IWPageForm* by click on *File -> New* and choosing *IntaWeb* tab. Click on the *Form* wizard and choose a *Page Form*. Once you have the form in the designer, drop a *TIWLabel* on it.

Make sure the form is named *IWForm1* (which corresponding to the *FormName* parameter in the IWP page). In the *OnRender* event, write the following code:

```
procedure TIWForm1.IWPageFormRender(Sender: TObject);
begin
  IWLabel1.Caption := FormatDateTime('ddd, dd of mmm, yyyy. hh:nn', Now);
end;
```

The only thing left now is to save the project and compile it. Once you have the executable ready, click on Deploying the files

15.4.6 Deploying the files

Deploying IWP is a very simple task. Other than configuring the web server to process IWP pages, the only other task is to copy the files to the correct location.

Independently of what web server you use, there are two things that need to be deployed: the static and dynamic pages is the first, and the actual web application is the second. The pages (independently of whether the are static or dynamic) need to be placed in the folder where your web server hosts web pages. In IIS this is normally *wwwroot* by default. In Omni it is *htdocs*.

The application needs to be deployed to the physical path where your virtual scripts folder points to. This must coincide with the *ApplicationURL* value entered in the *IWPageController*.

Calling the IWP pages
Once you have deployed all the files, all you need to do is call the page in question. For example, if your site is www.mydomain.com, and your IWP page is called first.iwp, then just open the browser and type http://www.mydomain.com/first.iwp.

The web server (via the association we made in Preparing the Server) automatically knows that when an IWP extension (or whatever extension you put) is requested, it will launch the corresponding web application (in our case *project1.dll*).

As you can see, IWP introduces a very easy yet powerful way of creating dynamic websites. There is hardly any code that needs to be written on the server side to handle the actual processing of the pages, other than the forms you wish to server dynamic pages with.

Section

16 IntraWeb Caching Mechanism

16.1 Caching Files

In order to improve performance, IntaWeb keeps files that can be cached (files from the *Files* directory etc) in a directory named *Cache.* By default this directory is created the first time the application saves a cached file in the root folder of the application. The user can change the default name and location of the cache directory, but in this chapter we will assume for the ease of explanation that it is named *Cache*.

There are several rules that apply to the *Cache* directory:

- the application should not save files for later use in the *Cache* directory, since the directory is emptied from time to time. The files in this directory are temporary files.
- when deploying an IntaWeb application, always make sure that it has full access (read, write, modify etc) to the *Cache* directory, otherwise the application will run with unpredictable results (access violations, internal errors etc).

Section

17 Deployment

17.1 Installation

17.1.1 Overview

Intraweb for Delphi

Applications created with Intraweb for Delphi are completely self contained. This means that you only need to copy the compiled output (.exe, .so, .dll, etc) to the server. You do not need to copy external files unless you have created external files.

Demos such as the Guess demo do not rely on any external files and thus simply copying the binary compiled ouptut to the server is sufficient.

Intraweb for Visual Studio .NET

Applications created with Intraweb for Visual Studio .NET must be deployed on machines that have the **1.1 .NET framework**. External files that the application can contained must be copied on the delpoyment machine as well.
deployment

17.1.2 External Files

It is possible that during the development the application created dependencies on external files. In this case you will need to copy these files as well.

Files
If you have files in the files directory or its subdirectories, you may need to copy those to the server as well. If so, they must resided in a *files* subdirectory that is placed in the same directory as the binary executable.

Templates
If you made use of templates, you will need to copy those to the server as well. They must resided in a *templates* subdirectory that is placed in the same directory as the binary executable.

17.1.3 Permissions

By default an IntaWeb application needs very few permissions to execute. These are the minimum requirements:

1. Read access to the *Files* if the files subdirectory is used.
2. Read access to the *Templates* subdirectory if templates are used.
3. Full access to the *Cache* subdirectory. By default if this directory does not exist, IntaWeb will create it and thus have the proper permissions. In this case, it must first have permissions to create directories in the directory it is contained in. You will get unpredictable errors, most probable access violations and internal server errors if access to this subdirectory is denied or restricted.

In addition if you are connecting to a database you will need to make sure that the application has proper permissions. This is especially true for local databases which access database files directly.

If using the BDE, you need to take special care so that the BDE can create its work directories as well as access the data files.

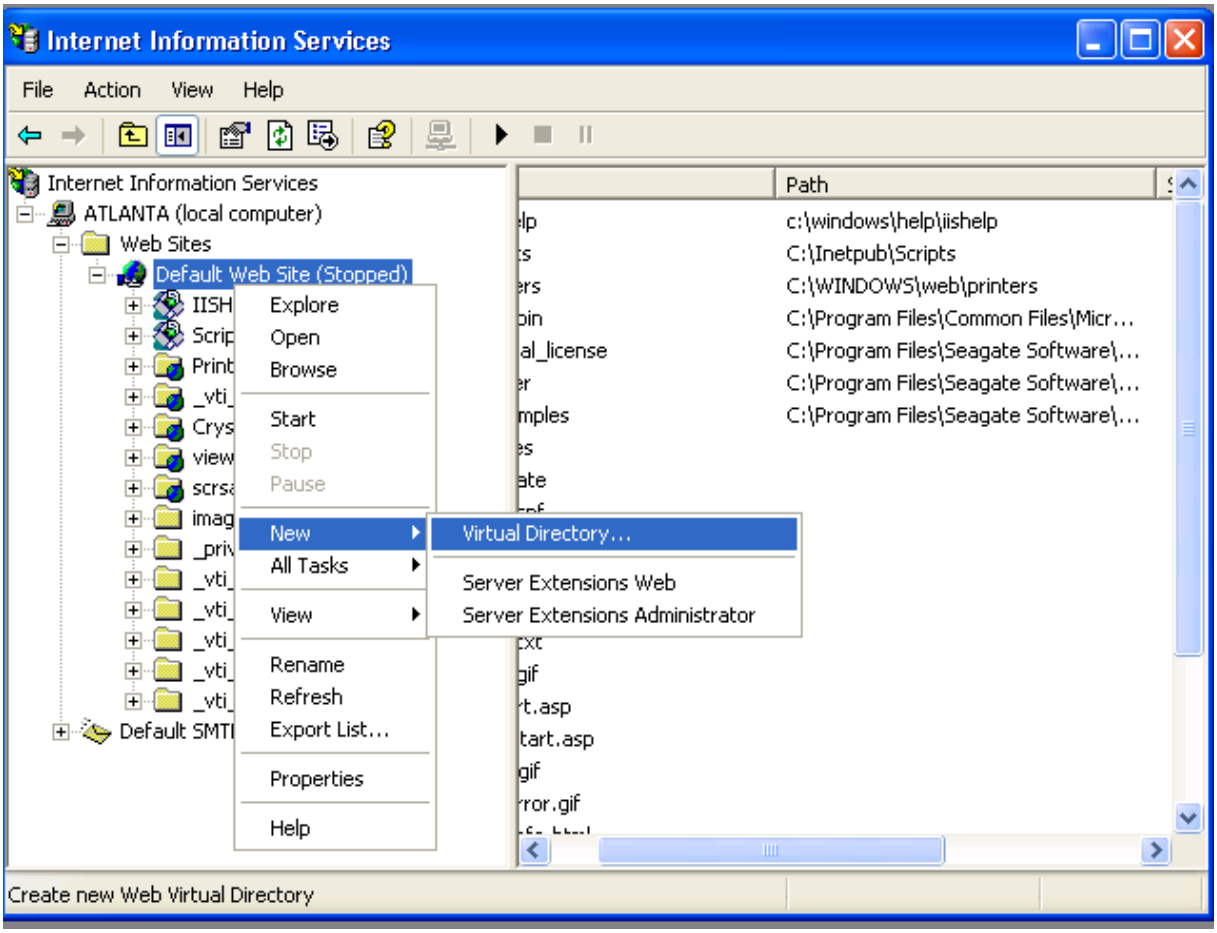
17.1.4 ISAPI

17.1.4.1 Deploying in IIS

Although an IW ISAPI application is the same as a standard ISAPI application, the steps to deploy an IW ISAPI will be explained step by step. However, due to the number of existing servers that support ISAPI's, this section is restricted to deployment on Microsoft's Internet Information Server. This is one of the most common servers to deploy ISAPI's on.

Before deploying the application, the first step is to configure IIS to allow ISAPI's to run. In IIS 5.0 which is the version included in Windows 2000 and Windows XP, the first step is to create the website and under this one create a virtual folder that has execute permissions to run ISAPI applications.
IMPORTANT: Please see note at the end of the page regarding IIS version 6.

Once the website is created (or on an existing website), right click with the mouse button and choose **New -> Virtual Directory**

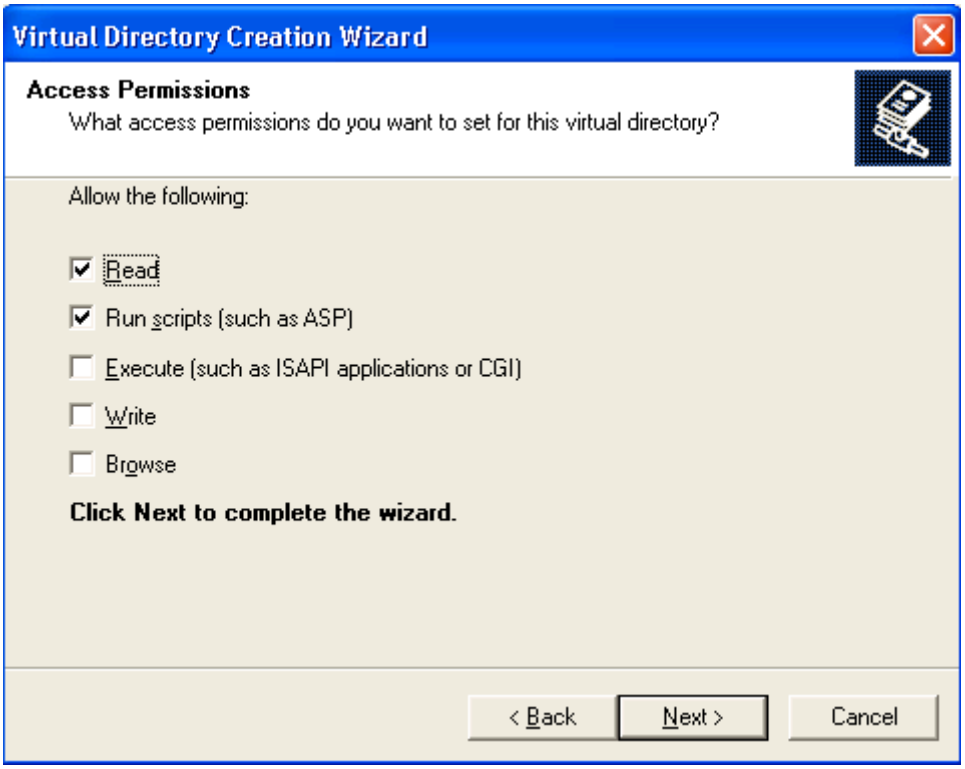


Click **Next** on the Wizard Introduction screen. The next step is to select an alias for this virtual directory. Normally, the standard is to use "scripts", however whatever alias desired can be used. This defines how the application will be called via the URL:

http://xxx.xxxx.xxx.xxx/{alias}/xxxxxxxxxx

The next step is to choose the actual physical directory where the files are going to reside and where the *alias* defined in the previous step will point to. Again, by default this would be something like

c:\inetpub\scripts. This of course depends on where IIS was installed and where the directory for the current website resides.



The last step is to choose what permissions the virtual directory is going to have. Normally, it is not a good idea to mix directories that have executables with directories that have files to be accessed for reading/writing. Actually, it is not recommendable to have any directories with write-access via IIS. In this case, the best option is to remove all permissions except the *Execute (such as ISAPI applications or CGI)*.

These are the steps required to configure IIS to run ISAPI's. However, before being able to run the application, there are a few more issues required. If the system is using NTFS, it is necessary to make sure that the physical directory that was entered in step 2 of the wizard and where the virtual alias points to, has execute permissions. Permissions for IIS are based on two factors, the ones assigned in IIS itself and those of the underlying file system, prevailing the latter over the former. Of course, if running FAT (highly an-recommended) there are no file level permissions.

The last step remaining is to copy the ISAPI into the appropriate physical directory that was setup, make sure that the ISAPI has execute permissions and copy any necessary files (such as **Files** and **Templates** folders) under the physical directory.

Once all these steps have been accomplished the application can be called using the URL:

http://xxx.xxx.xxx.xxx/{alias}/{application_name.dll}/{start_command}

Optionally if a different port than the default 80 has been configured in IIS, this value will need to be appended to the URL.

Example:
If the application runs on default port of IIS, the URL will look like this:

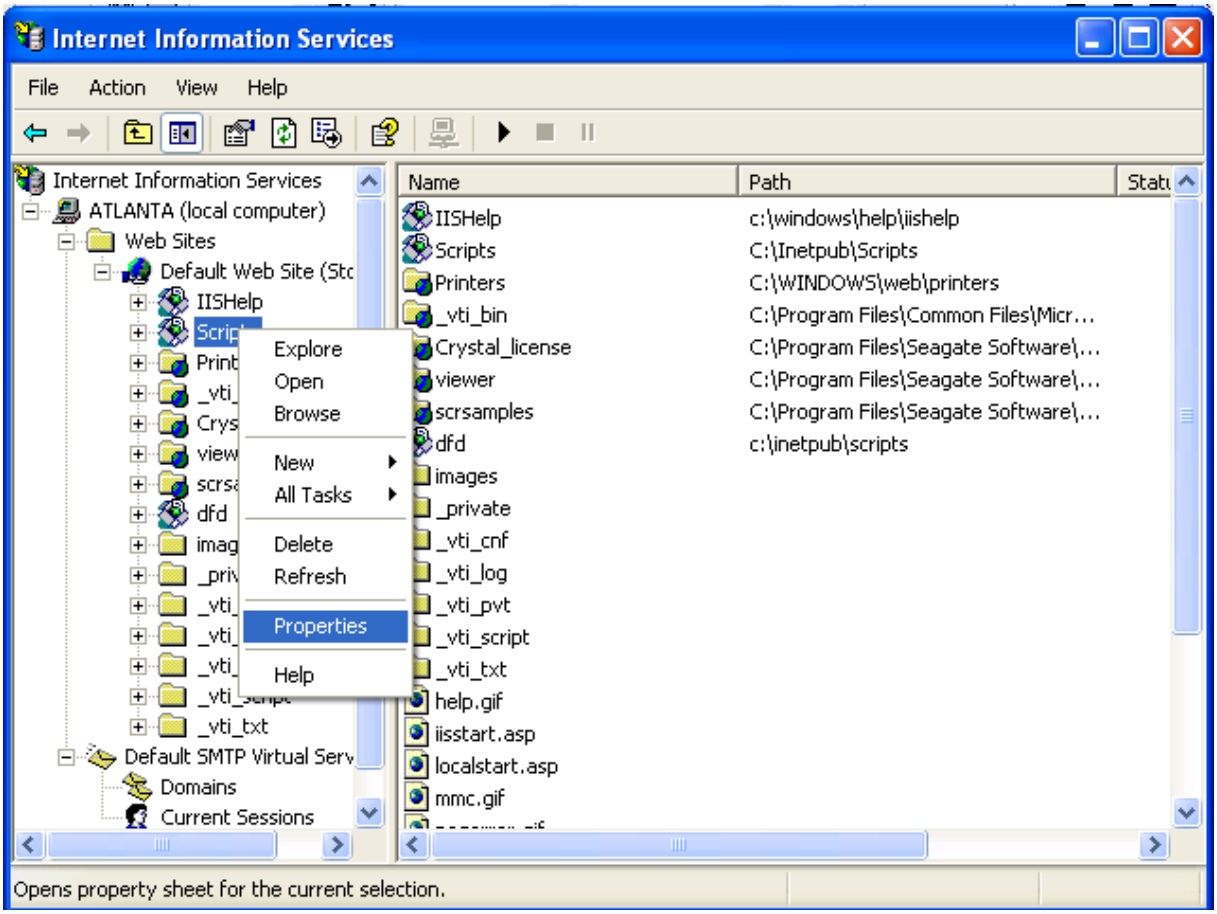
http://yourwebserver/virtualdirectory/YourDLLName/start

If you have setup your application to use a port the URL would look like:

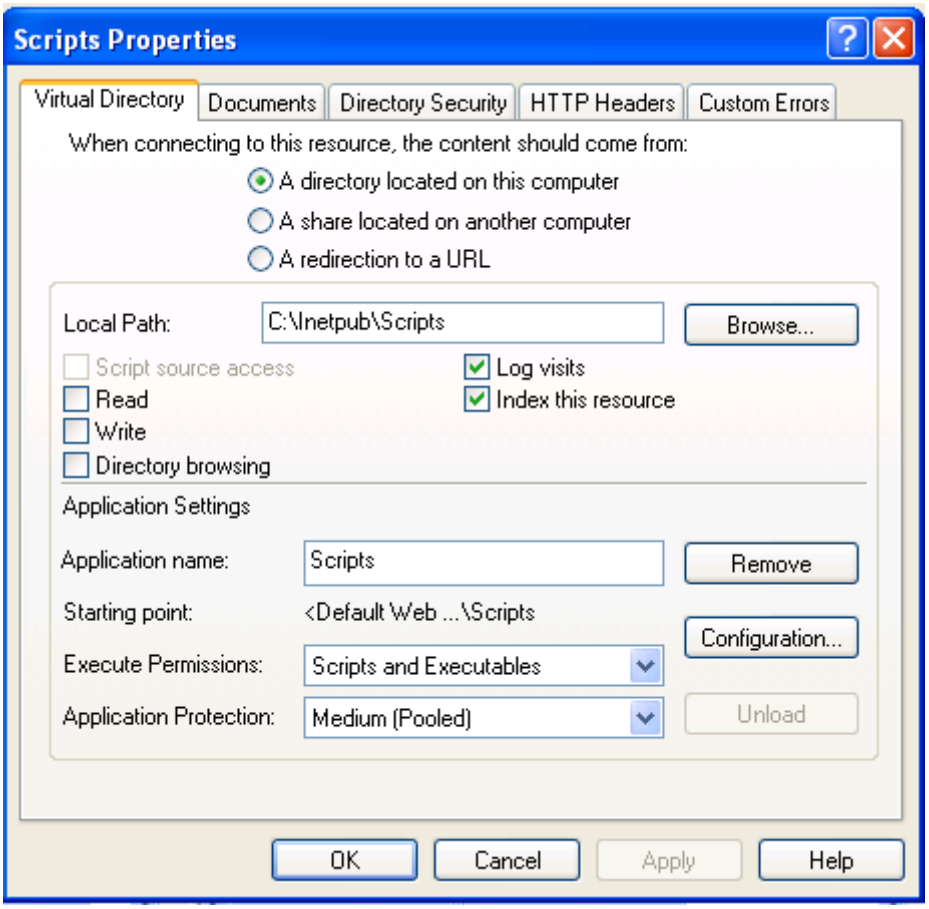
http://yourwebserver:port/virtualdirectory/YourDLLName/start

There is an important issue to remember when deploying ISAPI's under IIS (or for that matter under any other ISAPI compliant server), and that is that the ISAPI runs under certain security restrictions. In particular it runs under the context of a specific user, which in the case of IIS is IUSR_MACHINENAME, where MACHINENAME is substituted with the name of the server. This user has very restrictive security permissions and this has to be taken into consideration when certain calls require a higher level of security (such as communicating with COM objects, connecting to certain databases that require file access, etc).

As with any other ISAPI application, IW ISAPI's are DLLs and as such remain in memory after the first call. IIS allows them to be unloaded by configuring an application "area". To do this, right-click on the newly created virtual directory (in the figure below this is named *scripts*) and click on properties.



Once the property screen appears, click on the *Virtual Directory* tab to gain access to the *Application Settings*.



Enter a value for the *Application Name* (such as *Scripts* or *IW Applications*) and click on the *Create* button. Set the *Application Level* to the required isolation. Normally *Medium (Pooled)* is sufficient. For more information on isolation levels, consult the online help for IIS. When the application is accessed, the *Unload* button will be enabled and by accessing the properties and clicking on the *Unload* button, ALL ISAPI's located under the virtual directory will be removed from memory. Therefore, this only needs to be done ONCE per virtual directory.

NOTE:
IIS 6 has increased security by default. To enable ISAPI's to run, you need to click on the new entry "Web Service Extensions" in MMC and on the "All Unknown ISAPI extensions" change the status from "Prohibited".

17.1.4.2 ISAPI Utilities

These ISAPI management utilities can assist you with management of your ISAPI DLLs.

- DataWeb - http://www.turbodb.de/en/support/isapi.html
- EggCentric - http://www.eggcentric.com/

17.1.4.3 ISAPI Hosting

A list of hosting services which support ISAPI can be found on our website at:
http://www.atozedsoftware.com/IntraWeb/FAQ/Hosts.iwp

17.1.4.4 Common ISAPI Issues

ISAPI and thread pooling

The ISAPI and thread pooling section refers only to Intraweb for Delphi applications.
Starting with IntaWeb 5.1 the following export clause needs to be added into the project file (.dpr):

```
exports
  GetExtensionVersion,
  HttpExtensionProc,
  TerminateExtension;
```

This is necessary because of latest ISAPI and thread pooling modifications in IntaWeb.

ISAPI extensions stay in memory

For people not used to working with ISAPI, a common problem is "My ISAPI is loaded, and it won't unload, is this normal ?"
Yes, it is. The server alone can decide whenever to unload the extensions from memory. The unloading cannot be done from within the extension itself.
This design is intended to give a performance boost - opposed to CGI protocol, where the application is loaded every time it's executed. The ISAPI modules stay in memory, this reducing the time required to load. This system provides other advantages as well, like improved security and shared memory.

ISAPI extensions must be unloaded to be replaced

You may discover that, while developing, the executable (library) cannot be produced. This is a common situation when working with ISAPI and it's caused by the web server, which has loaded the ISAPI extension and still hasn't unload it. To continue your development you'll have to either unload the ISAPI manually, or rename the output file. For instructions on how to remove the ISAPIs manually, please see the documentation of the web server you are using.

ISAPI Manager

A solution to avoid the situation described previously is to use an intermediary DLL, which loads and unloads your application ISAPI as needed. We recommend to use ISAPI Manager, which can be downloaded from http://www.dataweb.de. Other tools that perform the same task are available.

How to unload from IIS

Apart from stopping and restarting the web server, IIS provides other means to unload the ISAPIs. This will be done automatically if you do the following:

For IIS 3 or lower:

To turn ISAPI caching off you will have to change the following registry entry to 0:
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\CacheExtensions
After changing this entry, restart the web server and ISAPIs won't be cached anymore.
To put the cache option back on, simply set the same value to 1 and restart the web server.

For IIS 4 or higher:

Right click on the script directory of your website, then select "Properties" and on the "Virtual Directory" tab choose the "Configuration..." button. Deselect the "Cache ISAPI applications" checkbox and click twice on "OK" to save the settings.

What does the Use ISAPIThreadPool checkbox mean ?

When creating a new ISAPI application, the wizard displays the option "Use ISAPIThreadPool". If checked, the unit ISAPIThreadPool unit will be included in the project.

According to Borland, for ISAPI apps the ISAPIThreadPool unit boosts scalability by using pooling threads and by handling a lot more users and concurrent requests.

This option should be used only when the applications runs under IIS.

17.1.4.5 Useful ISAPI links

For setting up IIS properly:
www.microsoft.com/directory/article.asp?ID=kb;en-us;Q300991&SD=MSDN

For building your first Delphi ISAPI:
http://www.matlus.com/scripts/website.dll/Tutorials?DelphiISAPI&FirstISAPI&3

For debugging Delphi ISAPIs:
http://www.msdelphi.com
http://codecentral.borland.com/codecentral/ccweb.exe/listing?id=15348
http://community.borland.com/article/0,1410,23024,00.html

17.2 Methods

17.2.1 Notes

Applications developed with IntaWeb for Delphi can be deployed as a Windows service / Linux daemon, a standalone executable, an ISAPI application, or an Apache DSO. Using page mode IntaWeb applications can be deployed by other methods as well.

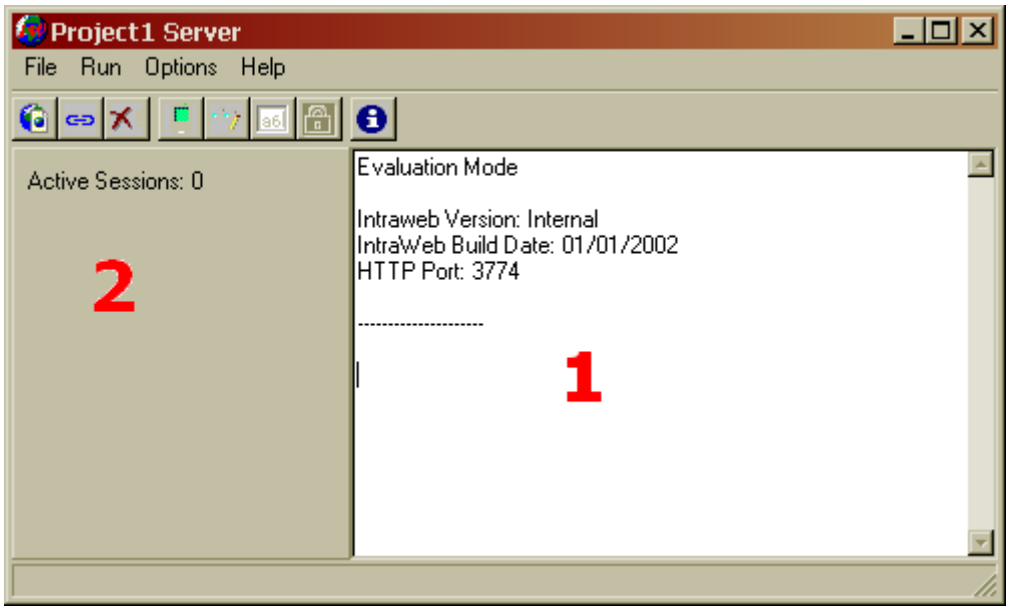
Applications developed with Intraweb for Visual Studio .NET can be deployed as standalone executables or as ISAPI dlls.

17.2.2 Stand Alone

While writing your application in Delphi, you will probably use the standalone mode to debug your application. When run as an application, a debugging screen appears with basic statistics. This screen also contains an Execute (Run | Execute or F9) option, which can be used to test execute the application in the browser. By clicking on it, the default browser will be launched with the corresponding URL to test the application.

The StandAlone debug form

The default window in StandAlone mode allows you to perform very basic debugging operations.



The label marked (2) in the picture displays the number of active connections to the current application.
The area marked with (1) in the picture represents the debug area.

To activate logging to the debug area, select "Options | Show Debug Information" from the menu.

Writing custom messages to the debug area

To write your custom messages in the debug area you may use the method "Log" of the StandAlone window. In Intraweb for Delphi, the default StandAlone window is of type TFormIWMain and is declared in the unit IWMain. In Intraweb for Visual Studio .NET the default StandAlone window is of type Main declared in the Atozed.Intraweb.NETMain namespace.

The "Log" method takes a parameter **AMsg** of type string, which represents the message to be written in the debug area.

Writing a custom StandAlone form

For a demonstration on writing your own custom StandAlone window in Delphi, please see the Custom StandAlone demo.

17.2.3 ISAPI / NSAPI / Apache DSO / CGI / Win-CGI

Intraweb for Delphi

Applications developed using page mode can be deployed as ISAPI, Apache DSO, NSAPI, CGI, or Win-CGI. Application mode executables may only be deployed as ISAPI, NSAPI or Apache DSO.

Intraweb for Visual Studio . NET

These kind of applications can be deployed only as ISAPI dlls.

17.2.4 Windows Service

Running IntaWeb applications as a service has its benefits and disadvantages. The disadvantages are that there is no debug screen or execute menu item. The main advantage is that the there is no requirement for logging on to the machine in order to run the application (like any other Windows service). A few steps have to be taken prior to running the application as a service. In particular, it has to be installed as such. To do so, using the windows command prompt, change to the directory where the application resides and type:

```
Application_name –install
```

This will install it and the application will appear in the Services Applet. From there, it can be configured to run automatically or manually. There is no need to activate the "Interact with Desktop" under the properties of the service, and doing so will have no effect whatsoever.

In a similar way, if the need arises to uninstall the application, it can be done by typing:

```
Application_name –uninstall
```

Before executing this command, be sure to stop your service.

Notes

1. Only Windows NT, Windows 2000, and Windows XP support services. Windows 95, Windows 98 and Windows ME do not support services.
2. Services do not function in evaluation mode. Attempts to do so will result in errors.

17.3 Launch URLs

17.3.1 Linking to IntraWeb Applications

Since the whole user interface is based on a web browser, calling the application is done using a URL. The URL has a very simple format.

Stand Alone Usage

Syntax: http://<server>:<port>
Example: http://www.atozedsoftware.com:4000

ISAPI Usage

Syntax: http://<server>/<script path>/<dll>
Example: http://www.atozedsoftware.com/iw/guess.dll

Apache DSO Usage

Syntax: http://<server>/<location>
Example: http://www.atozedsoftware.com/myapp

Launch URL Option

URL's are formed by specifying the host and adding the port number (if different to the default 80). You can optionally specify a start text on the URL by setting the StartCmd property in the ServerController and then adding the value of the URL For example:
http://www.atozedsoftware.com:4000/launch

17.3.2 Sessions

Every time this URL is entered into the browser and new session is created and the user is tracked automatically throughout the whole period that the session lasts. Optionally, parameters can also be

specified when calling a new instance by passing them using POST or GET.

17.3.3 Passing Parameters

Parameters are passed to the application using the interrogation (?) sign after the start URL. Each parameter consists of a "parameter name" and "parameter value". Parameters are separated from each other using the ampersand (&) sign. The following examples show how to pass two parameters named "param1" and "param2" with values "value1" and "value" respectively (Example is for standalone):

```
http://www.atozedsoftware.com:4000?param1=value1&param2=value2
```

These parameters are available in your application by accessing the RunParams property of the TIWApplication object.
In addition, prelaunch changes can be performed in the ServerController.OnNewSession event. One such use may be to read the parameters that have been passed in an offer different users different starting forms.

IMPORTANT NOTE:

Some browser on Macintosh require the start parameter to have an additional / before the ?. For example, the following:

http://<server>:8888?param1=value1

might not work correctly on some Macintosh browser and should be changed to:

http://<server>:8888/?param1=value1

Since nearly all browser accept /?, there should not be side-effects of adding / before the ? in all calls.

17.4 Converting Project Types

17.4.1 Converting from Standalone to ISAPI in Delphi

Lets take the Guess demo as an example:

```
program Guess;

uses
  Forms,
  IWMain,
  Main in 'Main.pas' {formMain: TIWFormModuleBase},
  ServerController in 'ServerController.pas' {IWServerController:
TDataModule};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TFormIWMain, formIWMain);
  Application.Run;
end.
```

1. Change the `program` clause to `library`.
2. Remove the IWMain and Forms from the `uses` clause.
3. Add IWInitISAPI to the `uses` clause.
4. Remove everything between BEGIN..END.
5. Add IWRun between BEGIN..END.
6. Add GetExtensionVersion, HttpExtensionProc, TerminateExtension to the `exports` clause.

Your program should now look like this:

```
library GuessDLL;


uses
  ISAPIApp,
  IWInitISAPI,
  Main in 'Main.pas' {formMain: TIWFormModuleBase},
  ServerController in 'ServerController.pas' {IWServerController:
TDataModule};

{$R *.RES}
exports
  GetExtensionVersion,
  HttpExtensionProc,
  TerminateExtension;

begin
  IWRun;
end.
```

17.4.2 Converting from StandAlone to ISAPI in CBuilder

Open your bpr file in a text editor such as NotePad.

1. Change the EXE to DLL
2. Locate the line that says **-tW -tWM"/>** and add "D" after the first W
3. Change c0w32.obj to c0d32.obj
4. Save the changes and open the updated project file in C++ Builder
5. Remove IWMain and Forms from the uses clause
6. Add IWInitISAPI to the uses clause.
7. Build the project

17.4.3 Converting from ISAPI to StandAlone in Delphi

Lets take the GuessDLL demo as an example:

```
library GuessDLL;


uses
  IWInitISAPI,
  Main in 'Main.pas' {formMain: TIWFormModuleBase},
  ServerController in 'ServerController.pas' {IWServerController:
TDataModule};

{$R *.RES}

exports
  GetExtensionVersion,
  HttpExtensionProc,
  TerminateExtension;

begin
  IWRun;
end.
```

1. Change the `library` clause to `program`.
2. Remove the IWInitISAPI from the `uses` clause.
3. Add IWMain and Forms to the `uses` clause.
4. Remove everything between BEGIN..END
5. Add the lines below between BEGIN..END
6. Remove the `exports` clause.

Your program should now look like this:

```
program Guess;

uses
  Forms,
  IWMain,
  Main in 'Main.pas' {formMain: TIWFormModuleBase},
  ServerController in 'ServerController.pas' {IWServerController:
TDataModule};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TFormIWMain, formIWMain);
  Application.Run;
end.
```

17.5 Additional Linux Information

17.5.1 Overview

IntaWeb applications which use TImage or TFrame will need the support of XWindows. Applications which do not use these two components do not require X Window support.

Installing a full blown X Server on a Linux server machine is not desirable or even possible in many cases. Instead you can run a special X server designed for web servers called X Virtual Frambe Buffer, or Xvfb for short. Xvfb is used by thousands of CGI and Apache DSOs.

Xvfb is included in many distributions such as SUSE. If you do not have Xvfb you can obtain it from one of the following URLs:

- http://www.xfree86.org/4.0.1/Xvfb.1.html
- http://chartworks.com/support/server/XvfbonUnix.html
- http://www.slac.stanford.edu/grp/cd/soft/unix/xvfb.html

Once it is installed you can use it by issuing:

```
Xvfb :99 &
```

No configuration is required.

Then before starting your IW app issue:

```
export DISPLAY=localhost:99.0
```

17.6 Application ports

When deploying the application, the application port should be chosen carefully.

Port numbers below 1024 are always reserved. Linux won't even let you the application them without permissions.
Under Windows, the operating system will let the application run on any port, and it's the job of the application to select an unused port.

It is in general a good idea to select a port superior to 1024 for your application. A port commonly opened by firewalls and configured in proxies is 8080.

Whichever port you chose, verify before that no other application is using it.

Section

18 Performance Evaluations

18.1 Tips

You will likely want to test the performance of your application. Many users test the performance improperly and thus receive misleading results. When testing be aware of the following items that can negatively impact your tests.

1. When using Internet Explorer, the first page will render quickly. However, when you click on a button or a link from the first page, Internet Explorer will then load extra libraries and cause a delay. This delay is caused by Internet Explorer and not the IntaWeb application. As you move to successive pages, you will notice that this delay no longer exists.
2. When using a browser on the same machine as the server the network is forced to use the "loopback" address. The loop-back address generally provides good performance however sometimes will introduce delays into the transfer of data.
3. When using a browser on the same machine as the server, the browser, network and application all compete for CPU, disk and memory at the same time. Most browsers are quite CPU and memory intensive, and thus negatively impact the server and your results.
4. When using Netscape and running your application from Delphi, the Delphi debugger hooks and Netscape conflict. Often you will have to task switch from the browser to the application to "unstick" the local network.
5. Anytime you run your server from within Delphi or Visual Studio .NET, their debuggers are active. The debugger not only consumes memory and CPU, but can also slow down the execution of your application. Under normal circumstances, this is perfectly acceptable, however keep this in mind if you are testing performance.
6. The first time you execute an ISAPI based application the web server must load the DLL and this will cause for a delay.

To properly test performance, you should run your application and browser on separate machines.

Section

19 Scaling IntraWeb Applications

19.1 Scaling Methods

What if my application grows too big for one server to handle? Can IntaWeb scale? Sure it can. IntaWeb can be scaled using a variety of methods. First also consider that in many applications you can handle more with less CPU because App mode is stateful. Without state, applications often spend a lot of CPU streaming state to a DB, reconstructing state, or needlessly rerunning queries against a database. That being said, there are still times you need to scale.

Add Another Tier
Use MTS, COM, SOAP, ASTA, Midas, whatever and split that piece into multiple machines. This will take processing out of the IntaWeb application and allow it to be distributed.

Beef Up You Database Server
Add more CPU power or memory to the database server. In many systems, the database is the bottleneck, and the web application spends the majority of its time waiting on the database.

Add More Memory to Application Server
Check your memory usage and make sure that your application server has the appropriate amount of memory. Virtual memory will be used if not enough physical memory is available, but this will slow down the response time and consume CPU cycles. Eliminating the use of virtual memory will increase efficiency and capacity.

Use a Multi-Processor Server
IntaWeb servers are fully threaded. Thus, IntaWeb servers will take advantage of multiple processors if present.

Distribute the IntaWeb Application Itself
If you have reached a level requiring you to scale the actual IntaWeb application, this can be accomplished as well. Please see Distributing the IntaWeb Application.

19.2 Distributing the IntraWeb Application

What will be presented here is not the only way that an IntaWeb application can be distributed, but it is the most common. It is very simple, and effective. This method can also be used in conjunction with the previously described methods.

Step 1 – Install Multiple Application Servers
Each server will need it's own IP address. For this example, we will simply refer to them as .1, .2 and .3.

Step 2 – Create a New DNS Record
Create a new DNS entry to identify the application. For this example, we will use iwapp.atozedsoftware.com.

Step 3 – Add Multiple IPs for the DNS Record
DNS allows multiple IPs to be assigned to a given record. In our example, we would assign .1, .2, .3 to iwapp.atozedsoftware.com. When multiple IPs are assigned to a single record, the DNS server will perform rotating DNS, sometimes also referred to as round robin. This means that the first request for iwapp.atozedsoftware.com will return .1, the second will return .2, the third will return .3, the fourth will return .1, and so on.

This will distribute the load across the servers. This method does not perform true load balancing, as it does not measure the load, it just distributes it. In most applications, the law of averages applies and it

is quite effective. If your application is such that it creates large imbalances, you will need to use a load balancing DNS server instead.

Step 4 – Create a Redirect Entry
On each application server create a redirect entry using the primary web server's configuration, or a page that performs a redirect to that servers actual IP. When the page or virtual entry is requested, the browser will not know that it has been redirected to an IP by the DNS server, as this is part of its normal operation. However, we must make sure that subsequent requests are routed to the same application server, as IntaWeb is stateful. Note that this only applies to Application Mode, and not Page mode. This step can be skipped for Page Mode.

The virtual entry or web page merely redirects the web browser to a URL containing its individual IP instead of iwapp.atozedsoftware.com. For example if our URL is http://iwapp.atozedsoftware.com, this entry might redirect the browser to . This URL demonstrates a stand alone IntaWeb application, but it can be adjusted to redirect to a static page, an ISAPI version, or a DSO version. The important thing is that the browser is redirected to the physical application server so each subsequent request will return to that server.

Section

20 Secure HTTP / SSL

20.1 Introduction

If your application is deployed as an ISAPI DLL or an Apache DSO, you need to use the hosting web server's SSL capabilities since it handles the HTTP protocol.

In Stand Alone mode, SSL is supported also. The first step is that you must obtain SSL certificates.

20.2 Enabling SSL

IntaWeb requires that your certificates are .pem format. To enable SSL support, follow these steps:

1. Download and install the SSL DLLs. Information on how to obtain the DLLs is available at http://www.nevrona.com/indy/ssl.html . The DLLs are free.
2. Set the SSLPort in the ServerController to a value other than 0. The default for SSL support is 443. If you are running a standard web server on the same machine and it supports SSL, it will already be using 443 and you will need to use another port.
3. Set the SSLCertificatePassword in the ServerController if you assigned a password to your certificates.
4. Place your certificates in the same directory as the application. The certificates must be named:
   - Cert.pem
   - Root.pem
   - Key.pem

If your certificates are not in .pem format, please see the section on converting to PEM format.

20.3 Converting Certificates to PEM Format

Converting Certificates to PEM Format
Chances are that your certificates were not delivered to you in .pem format. If they are not in .pem you must convert them for use with IntaWeb.

This procedure assumes that you have already received your key and certificate pair from some Certificate Authority (like Verisign or Thawte) and that you have them installed in Microsoft Internet Explorer in Personal Certificates Store.

Export Certificate
Select the certificate and export it as a .pfx file (Personal Exchange Format). You may optionally protect it with a password.

Convert .pfx to .pem
As part of the SSL download, a utility named openssl.exe was included. This utility will be used to convert your .pfx file.
To use openssl.exe, use the following format:
openssl.exe pkcs12 –in <your file>.pfx –out <your file>.pem
Openssl.exe will prompt you for a password. Enter it if you used one, or leave it blank if you did not specify one. It will also prompt you for a new password for the .pem file. This is optional, but if you protect it with a password be sure to fill in the ServerController.SSLCertificatePassword property in your application.

Splitting the .pem File
If you examine the new .pem file with a notepad, you will notice that it consists of two parts. The two parts consist of the private key and the certificate (public key) part. There is also some addition information included. IntaWeb requires that this information be separated into separate files.

Key.pem
Create key.pem with notepad and paste everything between and including these two statements:

-----BEGIN RSA PRIVATE KEY-----
-----END RSA PRIVATE KEY-----

Cert.pem
Create cert.pem with notepad and paste everything between and including these two statements:

-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----

Root.pem
The final file that IntaWeb requires is the Certificate Authority certificate file. You can obtain this from the Internet Explorer in Trusted Root Certificate Authority dialog. Select the Authority that issued your certificate and export it in Base64 (cer) format. This format is also the same as PEM format so after export simply rename the file to root.pem.

20.4 Example

An Intraweb for Delphi example with a test certificate is included and can be seen by examining the StandAloneSSL demo.

Index

- S -

step-by-step 102