IntraWeb Manual

Copyright Atozed Software
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 ................................................................................................................................. 10

**Part III Migration Guide** 12

1 Overview ................................................................................................................................... 12

**Part IV Installation** 16

1 Installation of License Keys ............................................................................................................ 16
2 IntraWeb for Windows .................................................................................................................... 16
   Uninstallation ................................................................................................................................ 16
   Installation ...................................................................................................................................... 16
3 Installing IntraWeb for Linux ......................................................................................................... 16
4 Upgrading to a new version ........................................................................................................... 16
   Delphi 7 Users ............................................................................................................................... 16

**Part V Demos** 19

1 Quick Start ................................................................................................................................. 19
2 Features ...................................................................................................................................... 19
3 Guess ......................................................................................................................................... 19
4 GuessWB ................................................................................................................................... 19
5 GuessMulti ................................................................................................................................. 19
6 FishFact ....................................................................................................................................... 19
7 FishFactDM ................................................................................................................................. 19
8 Phoenitics Customer Profiler ..................................................................................................... 20
9 StandAloneSSL .......................................................................................................................... 20

Copyright Atozed Software
Part VI Debugging 22
  1 Getting Started ................................................................. 22
  2 Debug Output ................................................................. 22
  3 Detecting Errors on Startup ............................................. 22
  4 Command Line Parameters ............................................. 22

Part VII Development 25
  1 Rethinking the User Interface ........................................... 25
  2 Writing Your First IntraWeb Application ......................... 25
  3 Images and Graphics ..................................................... 26
  4 Extending IntraWeb ....................................................... 27
  5 Working with COM ....................................................... 27
  6 Working with ClientDataSet Components ....................... 28
  7 Working with PDA ......................................................... 28
  8 Miscellaneous ............................................................. 29

Part VIII Form Management 31
  1 Working with Forms ..................................................... 31
  2 TFrame ........................................................................... 31
  3 Visual Form Inheritance ................................................. 31
  4 Managing Forms .......................................................... 31
  5 Form List ....................................................................... 31
  6 Showing Forms ............................................................ 32
  7 Hiding Forms ............................................................... 32
  8 Destroying Forms ........................................................ 32
  9 Passing Data Between Forms ......................................... 33

Part IX State Management 37
  1 Inherent State ............................................................... 37
  2 Restrictions ................................................................. 37
  3 Safe Storage ................................................................. 37
  4 Complex State and the Back Button ................................. 39
    Using the OnBackButton Event ....................................... 40

Part X Session Management 43
# Contents

3 IntraWeb and Websnap ................................................................. 66
   Creating the Demo ................................................................. 67
   Running the Demo ................................................................. 73

Part XVI Deployment ................................................................. 78

1 Installation .............................................................................. 78
   Overview .............................................................................. 78
   External Files ....................................................................... 78
   Permissions .......................................................................... 78
   ISAPI ..................................................................................... 78
      Deploying in IIS .................................................................. 78
      ISAPI Utilities .................................................................... 82
      ISAPI Hosting .................................................................... 82

2 Methods .................................................................................. 83
   Notes ..................................................................................... 83
   Stand Alone .......................................................................... 83
   ISAPI / NSAPI / Apache DSO / CGI / Win-CGI ..................... 83
   Windows Service .................................................................... 83

3 Launch URLs ........................................................................... 83
   Linking to IntraWeb Applications ......................................... 83
   Sessions ............................................................................... 84
   Passing Parameters ................................................................ 84

4 Additional Linux Information .................................................. 85
   Overview ............................................................................... 85

Part XVII Performance Evaluations .............................................. 87

1 Tips ........................................................................................ 87

Part XVIII Scaling IntraWeb Applications ..................................... 89

1 Scaling Methods ..................................................................... 89

2 Distributing the IntraWeb Application ...................................... 89

Part XIX Secure HTTP / SSL ....................................................... 92

1 Introduction ............................................................................ 92

2 Enabling SSL ......................................................................... 92

3 Converting Certificates to PEM Format .................................... 92

4 Example ............................................................................... 93

Index ......................................................................................... 0
“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
1 Welcome

IntraWeb is a revolutionary new way to create web-based applications. Built upon Web Solution Builder and earlier versions of IntraWeb, 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. IntraWeb does away with all these hassles and overheads. By simply creating the application using the component suite within Delphi, 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 IntraWeb and are 100% compatible. No HTML, CGI or JavaScript coding is required; all the coding is done using Delphi. 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 IntraWeb 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 IntraWeb Component Reference help file should be consulted when a reference for properties, events, methods and components is needed.

1.2 How IntraWeb Works

IntraWeb 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 IntraWeb application is similar to that of other web solutions such as ISAPI, CGI, or ASP.

IntraWeb 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 IntraWeb, the port numbers that your IntraWeb 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, IntraWeb applications only listen on the IP 127.0.0.1. No requests from other IP addresses will be answered.
No Services
IntraWeb 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.atozedsoftware.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 developer's 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.

**IntraWeb**
IntraWeb 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.

**Doc-O-Matic**
Doc-O-Matic 1.1 is used to build the IntraWeb Component Reference help file.
[http://www.doc-o-matic.com](http://www.doc-o-matic.com)

**Inno Setup**
The IntraWeb 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 IntraWeb.

**Indy**
IntraWeb 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](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/](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.

Thomas Brattli
http://www.dhtmlcentral.com
For eXperience DHTML coolMenus 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.

1.7 Requirements

1.7.1 Overview

The only requirement is that users of IntraWeb developed applications have browsers that are HTML 4 compatible *, since extensive use of HTML 4 and JavaScript are made. IntraWeb 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 IntraWeb version 4. Which we will continue to maintain and support. As of version 5.1, IntraWeb 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**
IntraWeb 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 IntraWeb 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. IntraWeb adjusts for these differences automatically. IntraWeb generates the appropriate HTML and JavaScript code for the browser. IntraWeb 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.

IntraWeb adjusts for these differences automatically. IntraWeb generates the appropriate HTML and JavaScript code for the browser. IntraWeb 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. IntraWeb performs all of this transparently to you and without using Java, ActiveX, or any plug-ins.

**Internet Explorer**
Internet Explorer versions 4, 5, and 6 are supported.

**Netscape 6**
Netscape 6 is supported. There are some known issues with Netscape 6 which are fixed in the recent GECKO engine used in Mozilla 1.0 RC1.

**Mozilla**
Mozilla is supported.

**Opera**
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. Work is being done to provide full support for Opera with the next versions of Opera and IW. Other functionality IS supported in Opera.
1.7.3 Development Environments

When purchasing IntraWeb 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 (Pro and Enterprise)
- C++ Builder 5
- C++ Builder 6
- Kylix 2
- Kylix 3

There is also a Java edition that supports JBuilder, which is sold as a separate product.
Section II
2 What's New

2.1 Overview

IntraWeb 5.1 introduces new features and enhancements in the following areas:

- New Service Application Template
- Project source files for Standalone applications follow the Delphi standard DPR format
- Files directory now optionally supports subdirectories. For this functionality to be enabled, the property must be set in the server controller.
- New Components:
  - TIWProgressBar
  - TIWMPEG
  - TIWMovie
  - TIWActiveX
  - etc.. (see the component palette for new components)
- Additional properties for existing components
  - Enabled property for TIWRadioGroup
  - BorderOptions property for TIWImage, TIWImageFile and TIWRectangle
  - AllowSubFolders for ServerController
  - Background color for TIWMemo
  - And much more....
- **PDA Support via HTML 3.2**
  - OnBackButton: allows the developer to assign an event when a form that has been already posted is re-posted.
  - More custom error handling. Customize the following events:
    - Unknown Browser
    - No JavaScript support
    - No Cookie support
  - Session tracking now supports hidden fields (tmHidden)

and much more......

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, IntraWeb allows you to develop robust and efficient applications for any HTML 3.2 compliant PDA device.
Section III
3 Migration Guide

3.1 Overview

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.

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 sepaate 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:

```pascal
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:

```pascal
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):

```pascal
library GuessDLL;
uses
  IWInitISAPI,
  Main in 'Main.pas' (formMain: TIWFormModuleBase),
  ServerController in 'ServerController.pas' (IWServerController: TDataModule);
{$R *.RES}
begin
  IWRun;
end.

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 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 the Borland Community Site.

**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:

```pascal
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 is present for new projects:

```pascal
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* 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.
Section IV
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 IntraWeb program group.

4.2  IntraWeb for Windows

4.2.1  Uninstallation

Uninstalling IntraWeb is done in the same way as with other Windows applications. Select Add/Remove Program from the Control Panel and click on IntraWeb to remove it from the system.

4.2.2  Installation

The installation will automatically integrate IntraWeb into Delphi. Three new tabs will be created on the component palette containing the IntraWeb components. One of them contains the non-database components, another contains the data-aware ones, and the third contains control components. A new tab will also be created in the Delphi repository. All IntraWeb applications should be created using the templates contained in the repository under the tab IntraWeb.

The installation process copies the appropriate files to the Windows\System directory and to sub folders of all the Delphi environments selected. It also creates program group, which can be accessed via the Start menu. The documentation is placed in this program group.

4.3  Installing IntraWeb for Linux

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

4.4  Upgrading to a new version

Before upgrading to a new version:

1. Uninstall previous versions of IntraWeb
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 IntraWeb

4.4.1  Delphi 7 Users

If you are using the version of IntraWeb 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 IntraWeb:
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 IntraWeb with a simple application. After that you can then look at the other demos to see more advanced features of IntraWeb.

Many of the demos 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 IntraWeb merely to demonstrate them.

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

5.2 Features

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

5.3 Guess

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

5.4 GuessWB

GuessWB is the Guess application implemented using page mode instead of application mode.

5.5 GuessMulti

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

5.6 FishFact

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

5.7 FishFactDM

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

Phoenitics 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.

5.9 Stand AloneSSL

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

5.10 WebSnapSurvey

WebSnap survey shows how to use IntraWeb 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.

5.11 Custom Stand Alone

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

5.12 Back Button

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

5.13 Page Forms

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

5.14 Fish Market

Fish Market 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.
Section VI
6  Debugging

6.1  Getting Started

When using standalone mode debugging is the same as any other Delphi 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 screenshot with the debug tool bar button depressed, and debug output information from one user session:

![Screenshot of Debug Tool]

6.3  Detecting Errors on Startup

If errors occur during start up of an application IntraWeb 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 edlin.

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.
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 wouldn't it? 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 many 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

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

This creates a framework for a new IntraWeb 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.

```pascal
program Project1;
```
uses
  Forms,
  IWMain,
  ServerController in 'ServerController.pas' (IWServerController:
  IWServerController),
  Unit1 in 'Unit1.pas' (IWForm1: IWFormModuleBase);
{$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 IntraWeb application. Like
any other Delphi project, a main IntraWeb 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.

Granted what we have shown in this example is not rocket science. But, it has been created using
standard Delphi code and without any HTML. The example presented has been chosen because of its
simplicity. The purpose is to demonstrate that programming IntraWeb applications is very much the
same as developing any other Delphi application. The same methods, properties, events, etc can be
used in the same way. However, IntraWeb 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

7.3 Images and Graphics

IntraWeb supports graphics via the use of templates, TIWImage, TIWDBImage and TIWImageFile.
There are many ways to use graphics with in IntraWeb, 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
TIWImage 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**

TIWDBImage 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**

TIWImageFile 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**

IntraWeb can support GIF files however the install does not install GIF support. Please see the IntraWeb FAQ for details on how to use GIF files with IntraWeb.

### 7.4 Extending IntraWeb

**Custom Components**

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

IntraWeb 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 initialization is called. This is the default.
- **ciNormal**: COM is initialized. This is the method used when deploying Standalone applications built with IntraWeb.
- **ciMultiThreaded**: Use this setting when requiring COM initialization and when deploying as an ISAPI DLL.

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

### 7.6 Working with ClientDataSet Components

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 `IWMMain`.

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

When `DBClient` is placed before `IWMMain` IW will free sessions before `DBClient` interfaces are freed.

### 7.7 Working with PDA

IntraWeb 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**

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.

**Using 3.2 controls**

PDA applications are restricted to controls that are of the class 3.2. There are two tabs on the component palette that have the corresponding 3.2 version component of the standard IW components. 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.

**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 TIWLayoutManagerHTML32 component and design your form using the editor. The second option is to use a TIWTemplateProcessor32 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.

**Setting the main form**
As with normal IW applications, PDA versions need to have a main form defined. To do this, use the SetAsMainForm32 method in the initialization section of the form you require to be the main form. This provides you with the flexibility of actually having ONE IW application serve both PDA's and non-PDA devices by defining a main form for each one.

**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.

### 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:

```html
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.
Section VIII
8 Form Management

8.1 Working with Forms

The repository contains another unit, which is the IntraWeb Form. All new IntraWeb forms should be created using File | New and choosing the IntraWeb Form. All forms used by an IntraWeb application must be an IntraWeb-specific form. Standard Delphi forms are not compatible with IntraWeb. A new unit and form will be created and displayed on the screen. Working with IntraWeb forms differs a little bit from working with standard Delphi forms. For instance, any form that is displayed must be done using the Show method of the form. In other words, ShowModal is not permitted or supported.

8.2 TFrame

Use of Delphi's TFrame is supported. For information on TFrame usage, please consult the Delphi help.

An example of TFrame usage with IntraWeb can be seen in the Features demo.

8.3 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 IntraWeb 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.

8.4 Managing Forms

Form management in an IntraWeb application is very similar to that of a normal Delphi application but with a few restrictions.

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. Forms must be owned by WebApplication.

8.5 Form List

IntraWeb 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 contains several methods for interacting with the form list and are documented in the help file.
8.6 **Showing Forms**

The general format to display a form is this:

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

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

```pascal
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`. One thing that is different however is that in a normal application the form is shown immediately when the `.Show` method is called. With an IntraWeb 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 IntraWeb. 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.

8.7 **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 IntraWeb 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.

8.8 **Destroying Forms**

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

In an IntraWeb 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 IntraWeb. 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 IntraWeb.
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.

8.9 Passing Data Between Forms

Data can be passed between forms 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 define two forms, TFormMain and TFormDialog. TFormMain is the main form and contains a button and an edit box. TFormDialog contains a memo field and a label.

When the user presses the button on the first form the text from the edit box will be added to the memo on the dialog form and the form will be displayed. The dialog form will also display how many times it has been displayed and allow the user to return to the main form.

Main Form – Design Time

![Main Form – Design Time](image)

Main Form – Source Code

```pascal
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;
```

```
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;

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.

**Complete Demo**
To see the project in action, please see the FormData.dpr project in the Demos directory.
Section IX
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. IntraWeb 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 application? What? You do not have to? EXACTLY! That is how you manage state in IntraWeb.

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 IntraWeb 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 IntraWeb application except as temporary storage under controlled circumstances. IntraWeb 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

Form / Datamodule Members
Since IntraWeb 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 IntraWeb project is created a shell user session object is created for you in the ServerController. The default ServerController looks like this:

```pascal
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.

9.4 Complex State and the Back Button

Many people quickly discover that when building an IntraWeb application the back button in the browser does not work. By default IntraWeb 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 IntraWeb 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 IntraWeb Systems

System not built in IntraWeb 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.

**IntraWeb 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

IntraWeb 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.
IntraWeb 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. Everytime 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:

\[
\text{ASubmittedSequence}, \ \text{ACurrentSequence}: \text{Integer}; \ \text{AFormName}: \text{String}; \ \text{var} \ VHandled, \ VExecute: \text{Boolean}
\]

- **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 IntraWeb 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 example of using the OnBackButton take a look at the BackButton demo located in the application folder.


10 Session Management

10.1 WebApplication Object

TIWApplication is to an IntraWeb 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.

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 IntraWeb classes.

WebApplication Property of a Control

The base IntraWeb 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 IntraWeb form or component, the threadvar version can be accessed. This is especially useful in global procedures, or in TFrame code.

10.3 Lifetime

A users session is automatically managed by IntraWeb. When a new session is started IntraWeb will create a new instance of a TIWApplication 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.

10.4 Implementation

Sessions are managed automatically by IntraWeb. Sessions are stored in memory on the server and therefore 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 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 Microsoft's ISA proxy server change IP addresses between HTTP requests for a given user and will cause multiple IP addresses to be seen by the IntraWeb 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 IntraWeb 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.

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.
10.8 Component Reference

More information on the methods and properties of the TIWApplication class is available in the IntraWeb help file.

10.9 How does Session Management Work?

When using Application Mode in IntraWeb, 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 IntraWeb accomplishes this and see how it can extend it to store certain information.

IntraWeb 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/YYYYYYYYYYYYYYY

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 an 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.
Section XI
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. IntraWeb has a base layout manager, TIWLayoutMgr that can be descended from to create new layout managers. Currently IntraWeb has two layout managers TIWLayoutMgrForm and TIWTemplateProcessorHTML. In the future, there will be other layout managers to support XML and more.

11.2 Form Layout

This is the default layout manager. If no layout manager is specified, and implicit TIWLayoutMgrForm will be created and used. TIWLayoutMgrForm creates HTML pages that have the same layout and look as the designed form.

11.3 HTML Templates

HTML Templates

Templates allow for advanced formatting and layout of individual forms. Templates also allow a web designer to design the layout of forms without using Delphi. 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 4 and JavaScript. Any framed controls will be rendered without frames when templates are used. If you wish to have them 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 TIWTemplateProcessorHTML component from the IntraWeb Control tab to your form.
2. Set the form's TemplateProcessor to the new TIWTemplateProcessorHTML component.

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% }. 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.

For components on a TFrame, HTMLName differs from Name. Because a TFrame is the owner of the components contained in it, components on a TFrame can have the same name as components on other TFrame instances, or as components on the form. To circumvent this components on a TFrame set their HTMLName to the frame name + component name at run time. For instance, if a component named Label1 is on a TFrame named Frame1, the components HTMLName at run time will be Frame1Label1. {%Frame1Label1%} is the tag that need to use when using components on a frame in a template.

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

11.4 System Templates

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

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%}

The Guess demo has these two templates implemented as an example.

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.

IWErro.html
System errors are errors that happen outside of your application and in the server portion of IntraWeb. 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 IWErro.html.
Section XII
12  Server Controller

12.1  What is the Server Controller?

Every time you create a new IntraWeb Application*, the project wizard creates a form called IWServerController. All IntraWeb 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 IntraWeb. 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 accesible via the Object Inspector. To view these properties, choose the IWServerController form from your project forms by hitting CTRL-F12.

For information on the properties of the ServerController, please see the IntraWeb component reference help file.
Section XIII
13 Writing Custom Components

13.1 Overview

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

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

13.4 Core Server

In each IntraWeb 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. The core server has methods that are of use to component writers, and is documented in the help file 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 IntraWeb Third Party program. Information about this program is available on the IntraWeb website. The IntraWeb Third Party program can help you with the development of your components, as well as the distribution and promotion of your components.
Section XIV
14 Javascript

14.1 Overview

IntraWeb 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 IntraWeb. IntraWeb 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 IntraWeb javascript libraries.

14.2 Areas of Implementation

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

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

14.3 Using ScriptEvents

Many IntraWeb visual controls have events associated to them such as the TIWButton.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.

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.

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 TIWEEdit 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:

```
alert("You have just entered this control");
```

Now when you execute this:

![IntraWeb Event Scripts Editor](image)

You will see the following alert:

![Microsoft Internet Explorer](image)
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:

```javascript
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(ASender)`

"ASender" represents the actual control. In a normal HTML page with controls, "ASender" would be "this". However, IW "proxies" the JavaScript functions and the parameter representing the object is "ASender".

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:

```javascript
ASender.style.color='white';
ASender.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:

```javascript
ASender.style.color='white';
ASender.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:

```javascript
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

IntraWeb 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 IntraWeb. It takes fours parameters:

```javascript
function SubmitClickConfirm(AObjectName, AParam, ADoValidation,
```

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 everytime 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.

DoSubmit
Section XV
15 Page Mode

15.1 Introduction to Page Mode

IntraWeb 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.

For more information on how to develop Page Mode applications, see Working with Page Mode

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 IntraWeb 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 -> IntraWeb -> New Form** a new Page Form is created. Before placing any code in either form, some previous steps are required.

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:

```pascal
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:

```pascal
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 IntraWeb can be integrated with WebSnap. A demo will be built that uses WebSnap to provide the framework, login, and session
management. IntraWeb 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. IntraWeb integration with WebSnap.
2. Use of IntraWeb Page Mode.
3. Use of WebSnap session management with IntraWeb.
4. Use of WebSnap for control of authentication.
5. Use of IntraWeb 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 IntraWeb 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 IntraWeb with WebSnap is to add a TIWModuleController. To simplify distribution and not require distribution of external files, IntraWeb serves "internal" files from its libraries. IntraWeb 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 IntraWeb requirements. This component can also be used to use IntraWeb with WebBroker and is demonstrated in the GuessWB demo that is provided with IntraWeb.

For the TIWModuleController to be effective, the application module also needs a TWebDispatcher. Open the application module and add a TWebDispatcher (from the Internet tab) and then a TIWModuleController (from the IntraWeb Control tab). No further changes are required, IntraWeb 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 IntraWeb 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 IntraWeb page form. Select File : New : Other : IntraWeb : Page Form. Save the form as Page1Form.pas. Now let's go back and link Page1Module to Page1Form. To do this create an OnGetForm event for the TIWPageProducer. The event needs to look like this:

```delphi
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 comboboxes 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.

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

```delphi
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 anyway';
          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;
        textMsg.Visible := textMsg.Lines.Count > 0;
        if not textMsg.Visible then begin
          RecordVote(LBest, LWorst);
          ProduceResponse := False;
          DispatchPageName('PageResults', WebContext.Response, []);
        end;
      end;
  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 IntraWeb 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 IntraWeb 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[Ord(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[Ord(High(i) – Ord(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:
<table>
<thead>
<tr>
<th>Title</th>
<th>Best</th>
<th>Worst</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Motion Picture</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The Wrath of Khan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The Search for Spock</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The Voyage Home</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The Final Frontier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The Undiscovered Country</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final Conflict</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insurrection</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Section XVI
16 Deployment

16.1 Installation

16.1.1 Overview

IntraWeb applications 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 output to the server is sufficient.

16.1.2 External Files

It is possible that during the development of your application that you have 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.

16.1.3 Permissions

By default an IntraWeb 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, IntraWeb 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.

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.

16.1.4 ISAPI

16.1.4.1 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.

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

![Internet Information Services](image)

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:

\[http://xxx.xxx.xxx.xxx/{alias}/xxxxxxxxxx\]

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 un-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.
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 DLL's 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.

16.1.4.2 ISAPI Utilities

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

- EggCentric - http://www.eggcentric.com/

16.1.4.3 ISAPI Hosting

A list of hosting services which support ISAPI can be found on our website at:
http://www.atozedsoftware.com/intraweb/hosts.html
16.2 Methods

16.2.1 Notes

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

16.2.2 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.

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

IntraWeb 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.

16.2.4 Windows Service

Running IntraWeb 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 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

2. Services do not function in evaluation mode. Attempts to do so will result in errors.

16.3 Launch URLs

16.3.1 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

### 16.3.2 Sessions
Every time this URL is entered into the browser and a 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.

### 16.3.3 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.
16.4 Additional Linux Information

16.4.1 Overview

IntraWeb 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

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
```
Section XVI
17 Performance Evaluations

17.1 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 IntraWeb 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, Delphi’s debugger is 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 XVII
18 Scaling IntraWeb Applications

18.1 Scaling Methods

What if my application grows too big for one server to handle? Can IntraWeb scale? Sure it can. IntraWeb 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 IntraWeb 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
IntraWeb servers are fully threaded. Thus, IntraWeb servers will take advantage of multiple processors if present.

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

18.2 Distributing the IntraWeb Application

What will be presented here is not the only way that an IntraWeb 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 its 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 server’s 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 IntraWeb 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 IntraWeb 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.
19 Secure HTTP / SSL

19.1 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.

19.2 Enabling SSL

IntraWeb 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 . 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.

19.3 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 IntraWeb.

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. IntraWeb 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 IntraWeb 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.

19.4 Example

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