uniGUI Developer's Manual

Table of Contents

Foreword 0

Part I Introduction 4

Part II Installation 6

1 System Requirements ... 6
2 Installation Instructions ... 7
3 Sencha Touch Installation ... 13
4 Running Demos ... 14

Part III Technology Overview 19

1 Unified GUI ... 20
2 Web Sessions ... 21
3 Deployment Options ... 22
4 Forms and Modules ... 24
Application MainForm ... 24
LoginForm ... 25
MainModule ... 27
ServerModule ... 27
ServiceModule ... 28
DataModules ... 28
Frames ... 29
5 Special Objects ... 30
UniApplication Object ... 30
UniSession Object ... 31
UniServerInstance Object ... 32

Part IV Developer's Guide 33

1 Creating a New uniGUI Application ... 33
Standalone Server Project ... 35
Standalone Server / ISAPI Module Project ... 35
ISAPI Module Project ... 37
Windows Service Project ... 37
2 Application Design Considerations ... 38
General Design Concept ... 38
Web Application Scalability ... 38
3 Web Deployment ... 39
Sencha License Considerations ... 39
uniGUI Runtime Package ... 40
Adjusting Paths ... 41
Standalone Server ... 44
ISAPI Module ... 45
IIS 5 ... 46
IIS 6 ... 51

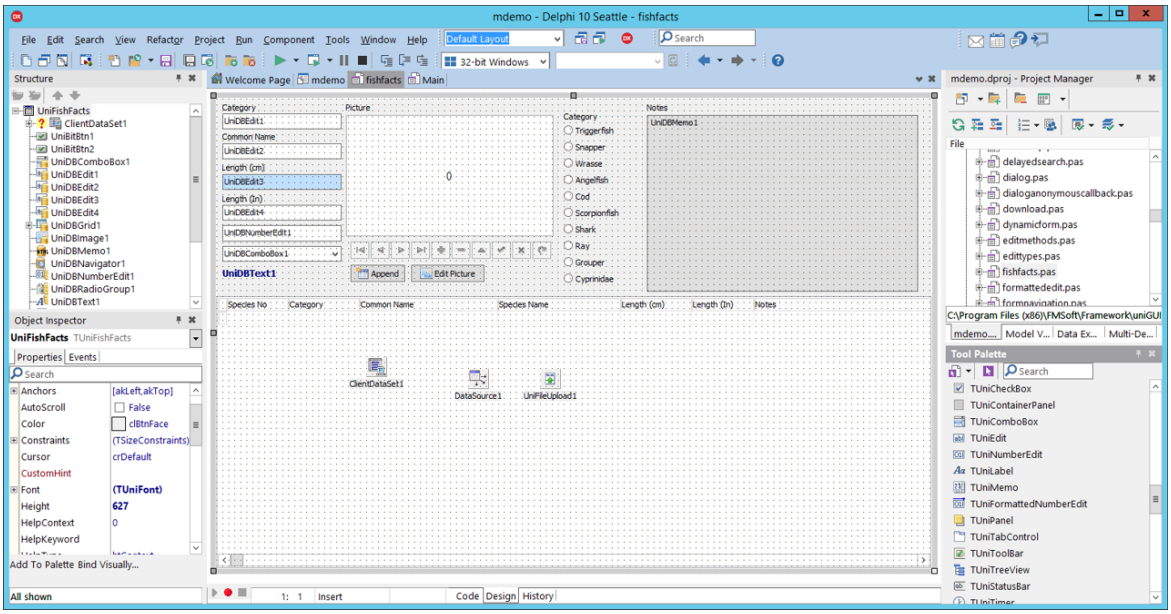
IIS 7 60
Apache 2.2 70
Windows Service 71
SSL Configuration 72
Obtain a SSL Certificate from an Authority 73
Generate a Self-Signed Certificate 74
Configure SSL Parameters 77
4 Stress Test Tool 79
Introduction 79
Usage 80
Recording a Session 80
Running the Stress Test 82
Server Flood Protection Considerations 85
Additional Settings 86
Index 87

1 Introduction

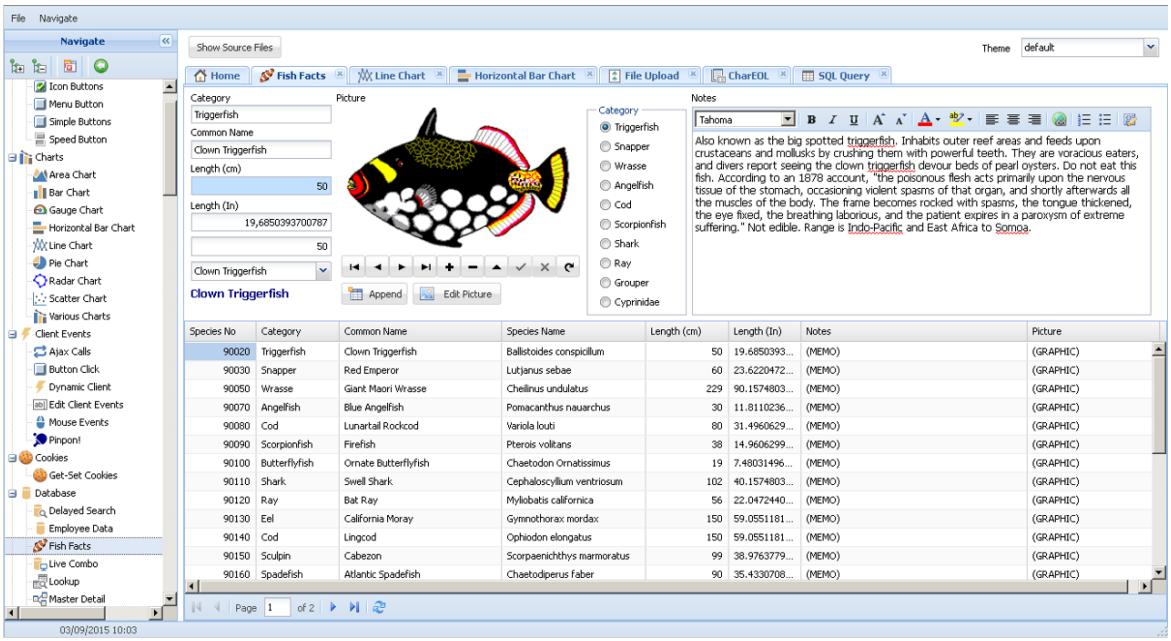
uniGUI is a Web Application Framework for Delphi. C++ Builder development is also supported for RAD Studio owners. **uniGUI** features a rich set of visual components to develop stateful Web applications which is very similar to developing a regular VCL application. uniGUI Web applications can easily be run and debugged directly in Delphi IDE which makes debugging process very easy and straightforward. **uniGUI** extends Web application development experience to a new dimension. In this new dimension productivity is the primary goal. **uniGUI** allows developer to focus on application business logic rather than working on web application development details such as working directly with HTML, JavaScript, XML Templates, and other web technologies. **uniGUI** will save many valuable development hours which in turn helps to considerably reduce project development, deployment and support costs. This makes **uniGUI** a perfect tool for small development teams which have a limited resources to spend on development details. uniGUI is also perfect for large teams who need to deliver enterprise scale projects in a limited time scale.

Deployment is another serious step in Web application development process. With uniGUI deployment is very easy and straightforward. Developers can choose one of the available options for deployment; Such as, **Windows Service**, **Standalone Server** or **ISAPI Module**. ISAPI modules can be deployed using **Microsoft IIS**, **Apache Web Server for Windows** or any other compatible ISAPI enabled Web server.

uniGUI uses Sencha Ext JS and Sencha Touch libraries for client side rendering. These libraries are considered as one of the industries leading frameworks to create **RIA** applications. uniGUI combines powers of Ext JS with RAD capabilities of Delphi which probably is the fastest way of creating a RIA applications in Delphi. uniGUI encapsulates Ext JS classes inside a special set of Delphi controls which enables developers to create feature rich web applications without a need to learn client side scripting and working on UI details on client side, so valuable development time can be spent on business logic rather than working on repetitive UI design details which can be time consuming and technically demanding tasks.



uniGUI application in Delphi 10 Seatle IDE



Web Application running inside a browser

2 Installation

2.1 System Requirements

Supported **Delphi*** versions are **Turbo Delphi**, **Delphi 2006**, **Delphi 2007**, **Delphi 2009**, **Delphi 2010, Delphi XE-XE8** and **Delphi 10 Seatle.**

C++ Builder is supported but not tested with all Versions.

uniGUI does not require any special hardware or OS configuration. A typical uniGUI installation will occupy 100-150MB of HDD space.

Note 1: Requirement for runtime environment can be very different and will be discussed under deployment topic.
Note 2: In order to install **uniGUI** for **C++ Builder** you need to have **RAD Studio IDE** installed. Installing **C++ Builder** alone is not enough. Please see Installation Instructions for details.

All **Delphi versions must be installed with latest available updates and service packs.*

2.2 Installation Instructions

Installation instructions for uniGUI (Delphi and C++ Builder)**

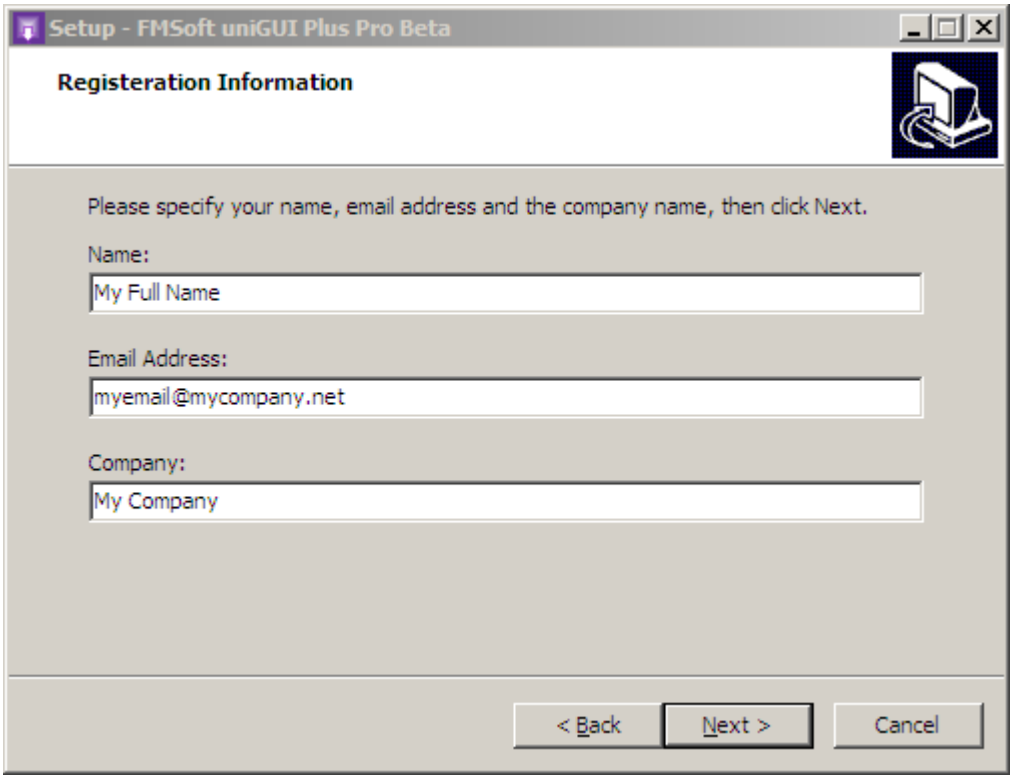
- Before installing a new version remove all design packages from Delphi and uninstall uniGUI from Windows Program Add/Remove.
- After re-compiling an application with this new version, "ext" folder must be re-deployed to PCs running new version of your application or you can simply re-install the newly introduced Ext JS runtime package which can be downloaded from Downloads page.

****Note for C++ Builder:** You need **RAD Studio IDE** to install **uniGUI** for **C++ Builder**.

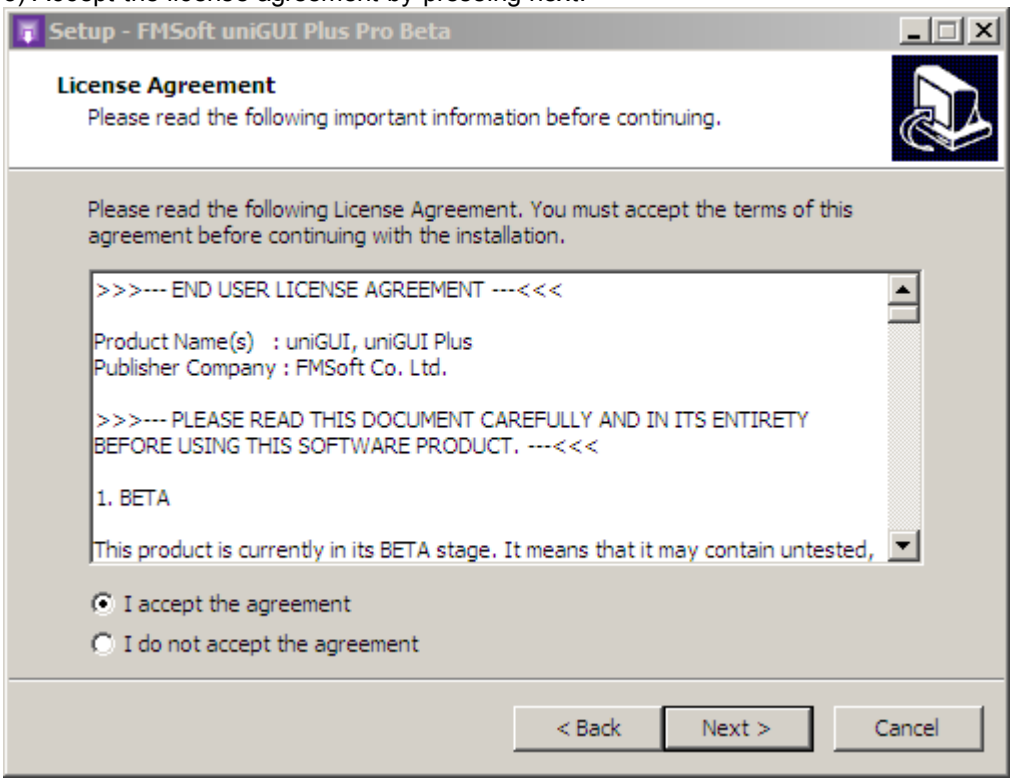
1) Download the latest **uniGUI** Setup from customer portal.
You will notice that there are two versions for of setup:

- **FMSoft_uniGUI_{Edition_0.XX.0.YYYY}_Beta.exe**
  - This is the one which will be installed on developer PC for development purpose only.
- **FMSoft_uniGUI_{Edition_runtime_0.XX.0.YYYY}.exe**
  - This one is only for deployment purpose and will be installed on your server PC which your uniGUI apps will run.

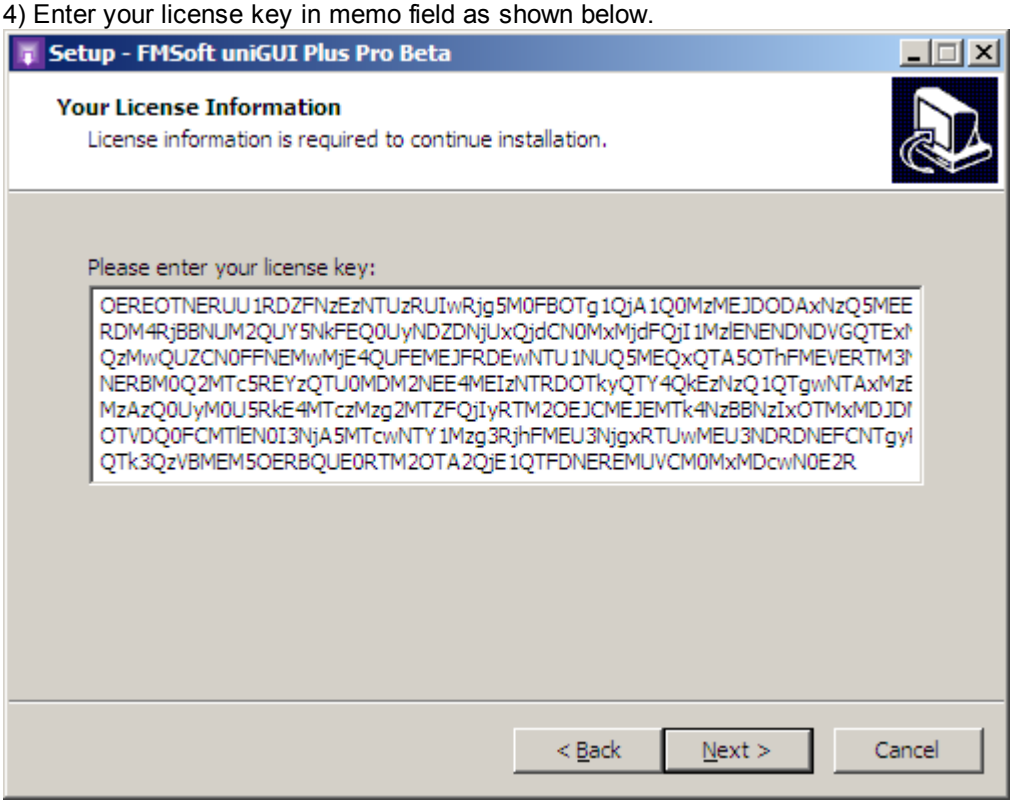
2) Enter required information. Make sure entered email address is same as email address registered in customer portal.



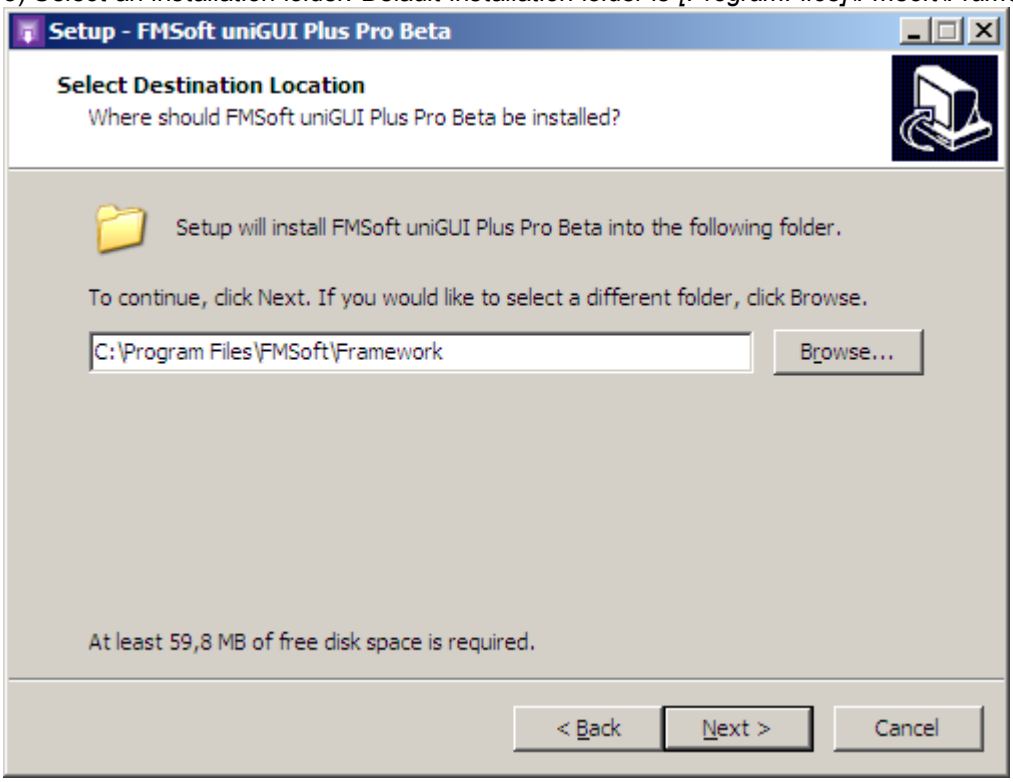
3) Accept the license agreement by pressing next.



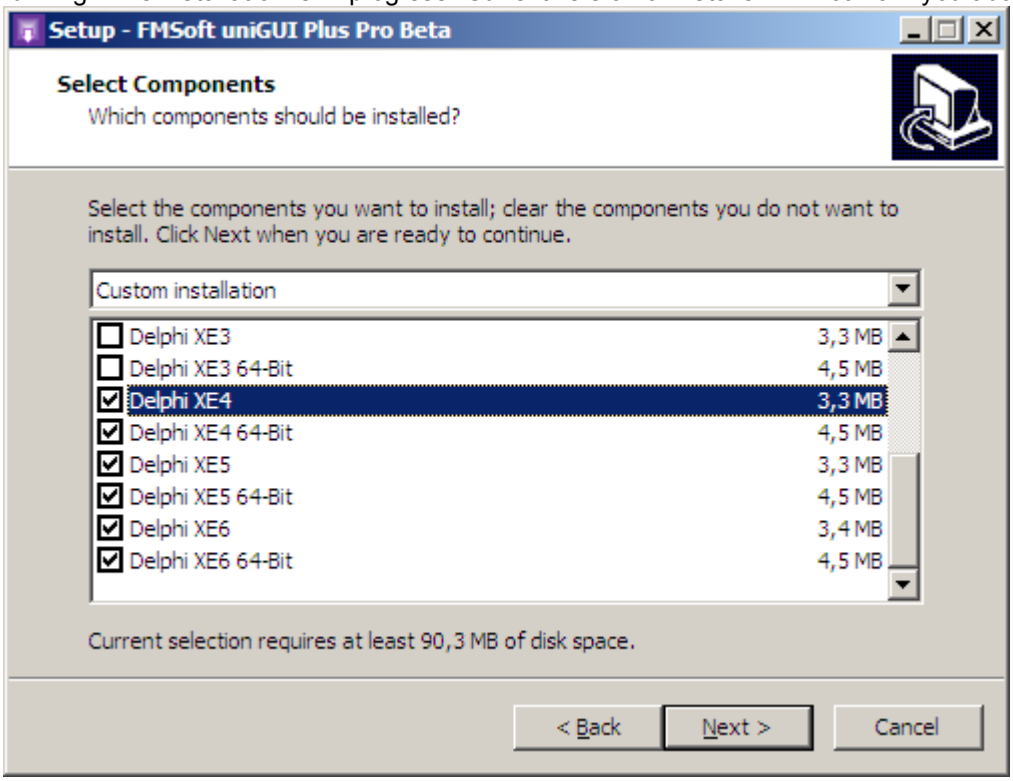
4) Enter your license key in memo field as shown below.



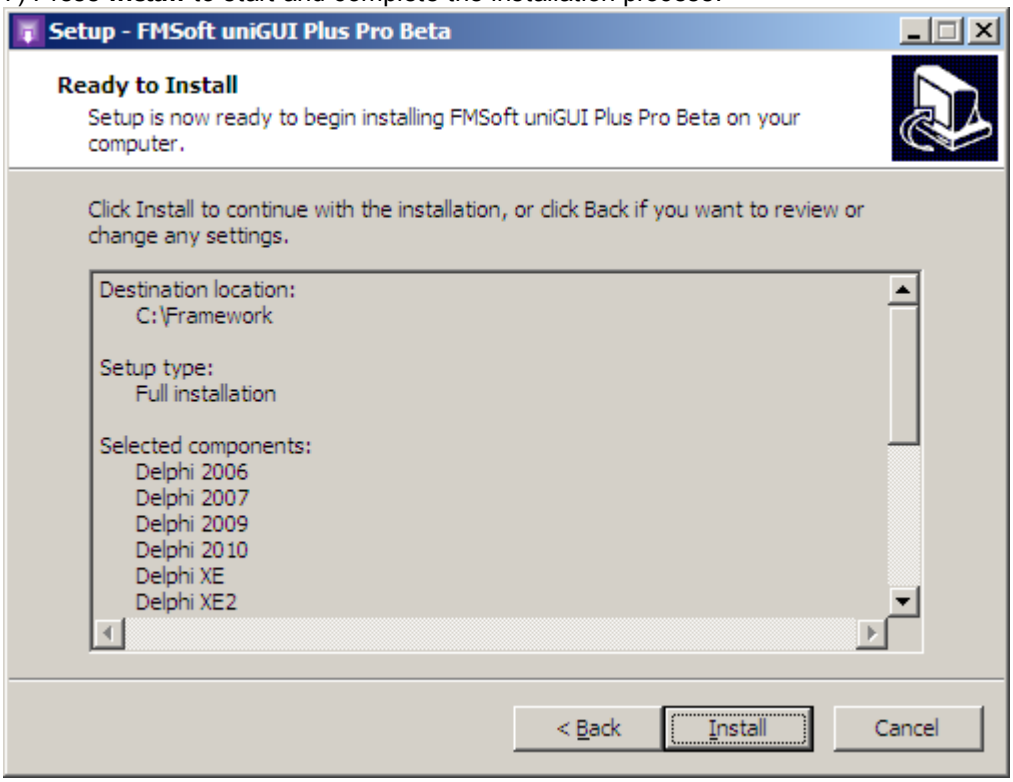
5) Select an installation folder. Default installation folder is *[ProgramFiles]\Fmsoft\Framework*\.



6) Select Delphi version(s) you want to install uniGUI library. You must be sure that Delphi is not running while installation is in progress. Current version of installer will not warn you about this.

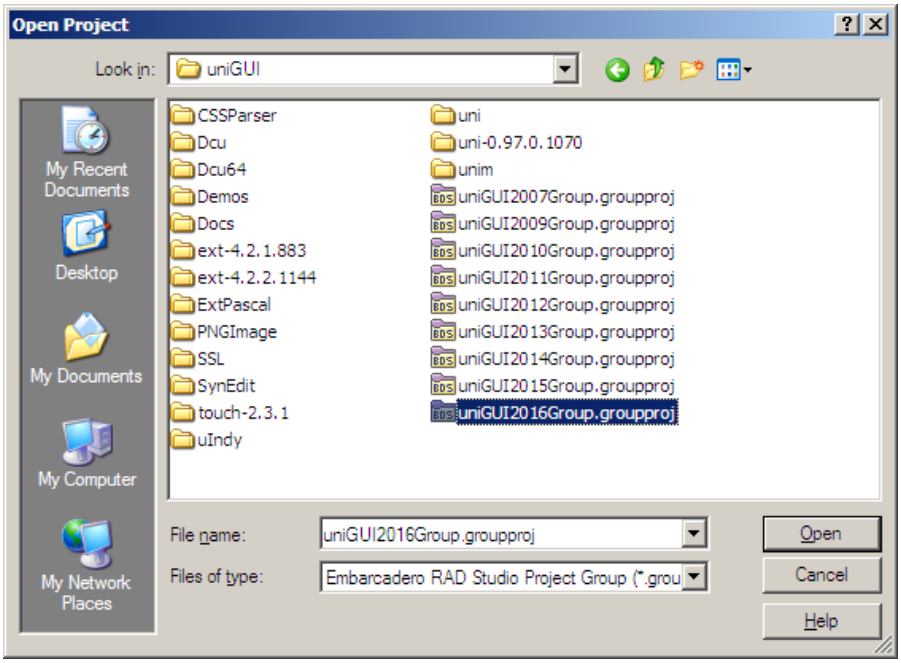


7) Press **Install** to start and complete the installation process.



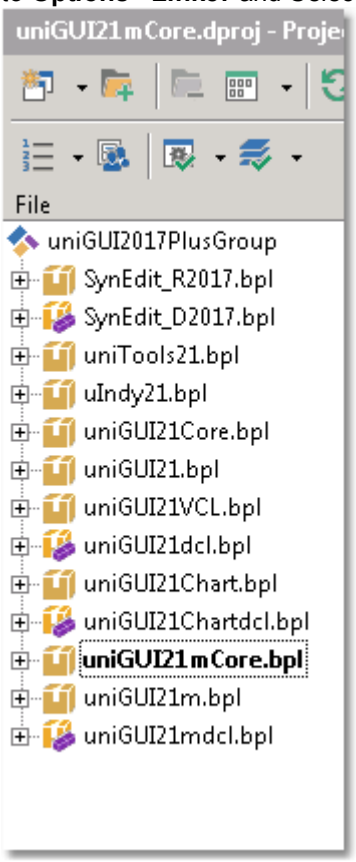
8) Start Delphi and open the project group for your Delphi version. e.g. **uniGUI2016Group (Delphi XE6).**

****Additional step for C++ Builder:** Instead of Delphi IDE open project in **RAD Studio IDE.**

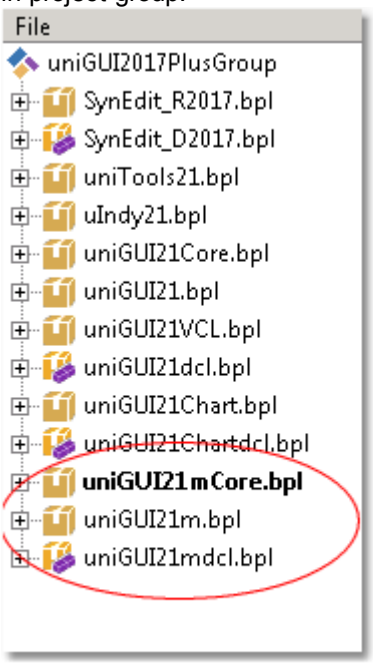


9) In project manager there are 11 Delphi packages. Build all packages starting from ***SynEdit_Rxxxx.bpl.***

****Additional step for C++ Builder:** Before building packages, for each individual package please go to **Options->Linker** and Select/Set **Generate all C++Builder Files**.



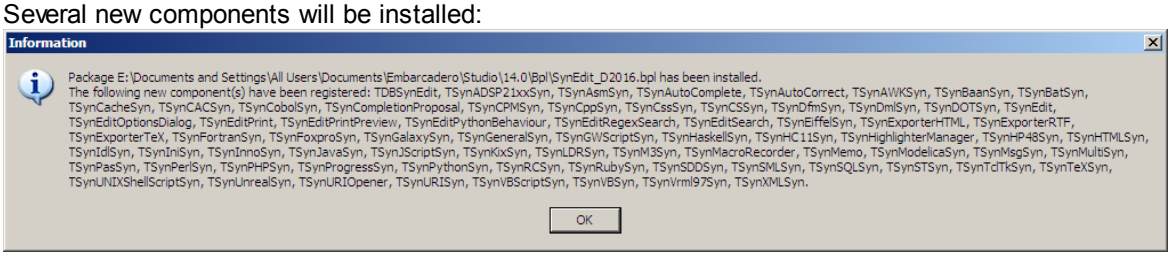
If you are installing the **Plus Edition** you will see three additional files related to Mobile development in project group:

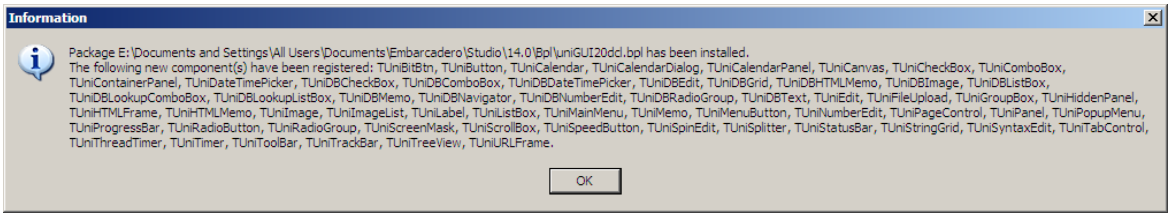


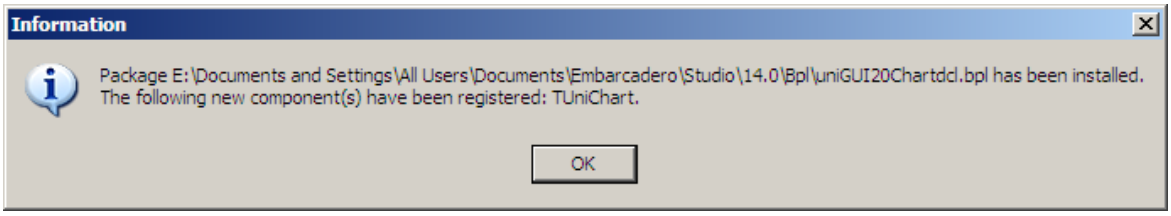
10) After building all packages install design time packages by right-clicking and selecting Install in following order:

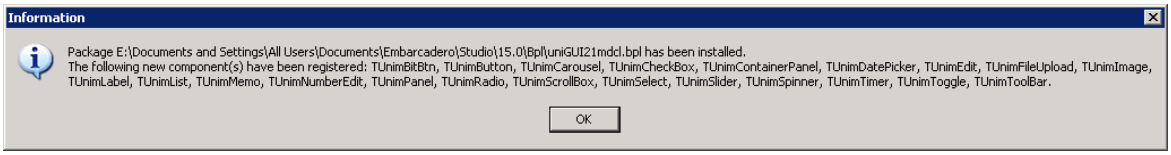
- **SynEdit_D20xx.bpl**
- **uniGUIxxdcl.bpl**
- **uniGUIxxChartdcl.bpl**
- **uniGUIxxmdcl.bpl** (Plus/Complete Editions only)

Several new components will be installed:









Plus Edition only

****Additional notes for C++ Builder:**

- After starting a new C++ project you must disable **Linker->Dynamic RTL**.
- Currently building with Run Time Packages doesn't work properly. You must statically link all libs and create a single EXE. (Simply unselect **Build with runtime packages** option.)
- New C++ projects are created without a resource (.RES) file. As a result project has no default Icon. This issue will be fixed in next releases.
- Combo VCL/ISAPI projects are not supported for C++ Builder.

2.3 Sencha Touch Installation

***Note:** This step is only required for **uniGUI Plus Edition**. For other editions you can skip this step.*

Current version of commercial **uniGUI Plus** with Sencha Touch support, does not distribute **Sencha Touch** files. (**uniGUI Complete** is the edition which deploys Sencha Touch.)

You must download it from Sencha Touch web site.

Here are the instructions:

Please download **Sencha Touch** from here. What you need from package is the folder named **touch-2.4.2** which will be extracted under **touch-2.4.2-commercial** folder.

You must copy **touch-2.4.2** folder to same folder your **Sencha Ext JS** folder resides.

For example if your folder structure looks like:

..\uniGUI\Dcu
..\uniGUI\Demos
..\uniGUI\ext-4.2.2.1144

You must copy **touch-2.4.2** folder to **..\uniGUI.**

So after copying it should look like below:

..\uniGUI\Dcu

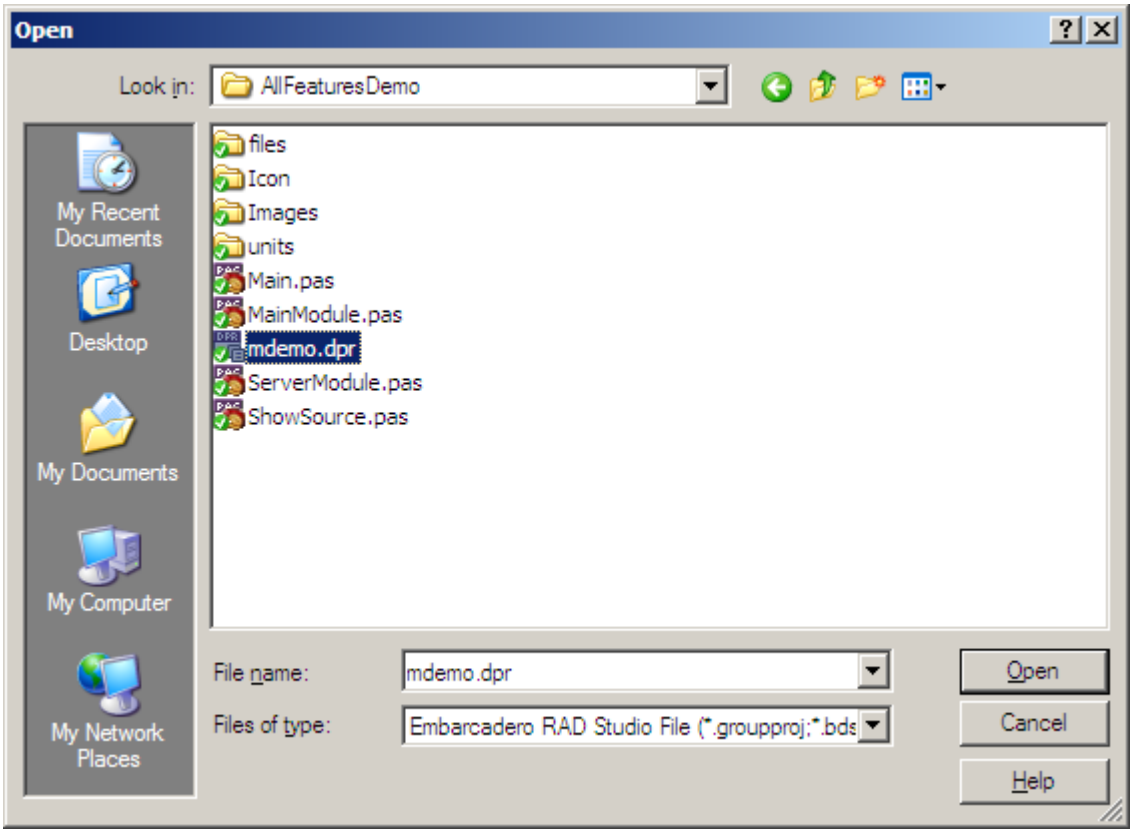
..\uniGUI\Demos
..\uniGUI\ext-4.2.2.1144
..\uniGUI\touch-2.4.2

2.4 Running Demos

Demos are a good place to start and become familiar with framework's basics quickly. Fortunately uniGUI comes with plenty of example projects which can fully demonstrate all aspects of uniGUI including all standard and advanced features. Almost all of uniGUI demos are compatible with all supported versions of Delphi. A few demos may not compile on earlier versions of Delphi because of language incompatibilities and lack of new features introduced in newer versions of Delphi. Running demos is a very simple task and quite identical to run any VCL application except uniGUI application will run inside a browser.

In order to run a demo in Delphi IDE go to File->Open and point location to where uniGUI demos are located. Normally under *<install folder>\FMSoft\Framework\uniGUI\Demos\...*

Select demo folder named *AllFeaturesDemo* and open *mdemo.dpr*.



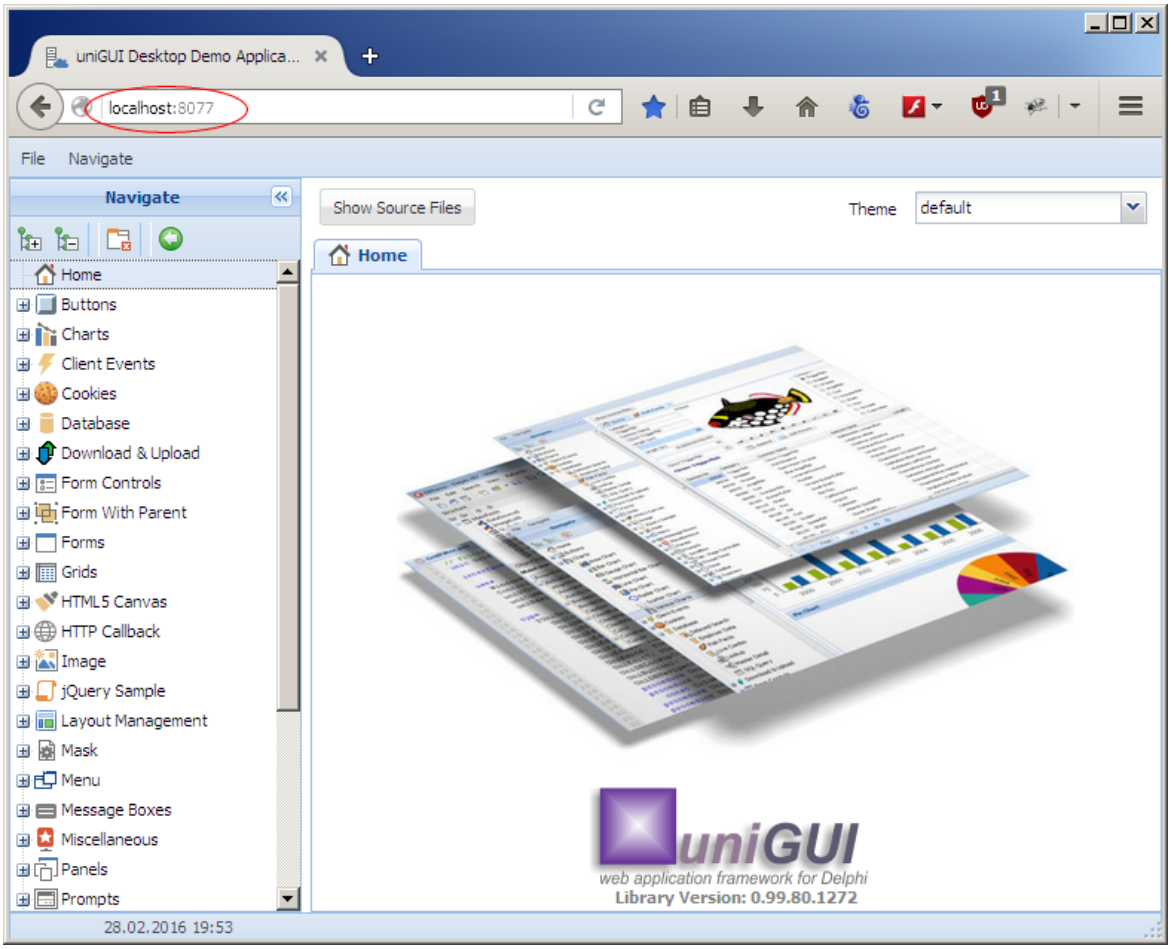
In IDE select *Project->Build mdemo* and build the project. After a successful build press Run and you will notice a new icon will appear in tray 

At this stage a firewall warning may appear in desktop asking for permission to allow uniGUI application to listen its dedicated port. Please grant required permission to firewall.

Now in order to run a web session please open one of compatible browsers such as FireFox, Chrome, IE9+, Safari or Opera.

In Browser address bar type below address:

http://localhost:8077

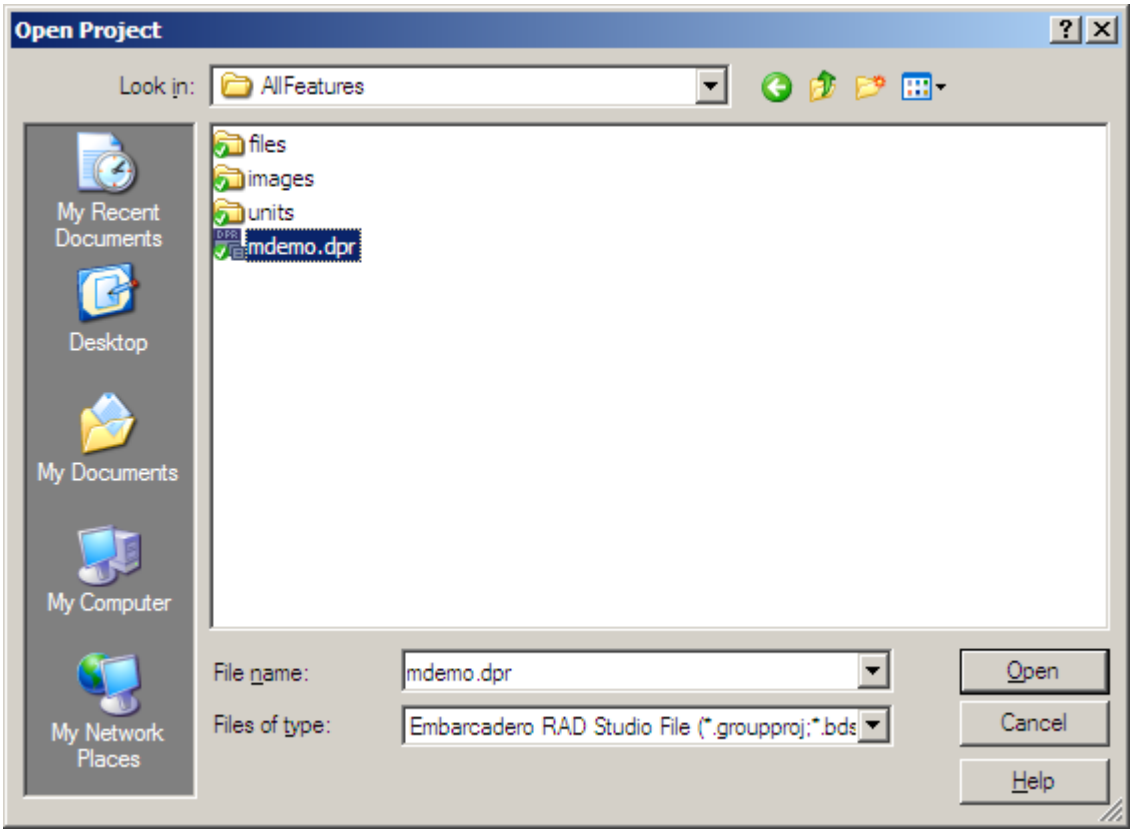


uniGUI application running

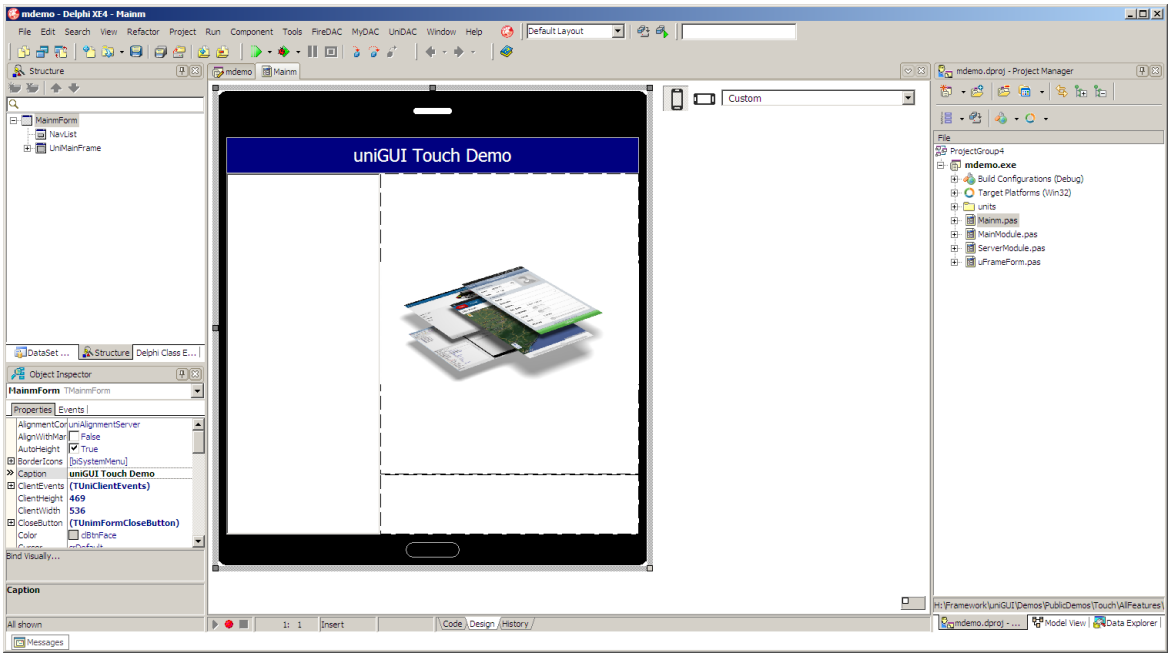
You can terminate running application server bu right-clicking on tray icon and selecting shutdown from pop-up menu.

If your uniGUI edition contains support for mobile application you can run demos you can follow same principles to run mobile application demo.

This time please select and open *mdemo.dpr* from ..*Touch\AllFeatures* folder.



You can open Mainm.pas in IDE to see how mobile designer looks in IDE. It looks a bit different than standard Form designer and modified to reflect look and feel of a mobile device.



uniGUI Mobile web application in IDE

Now you can run application server by building project and running it.

Again a new tray icon will appear. 

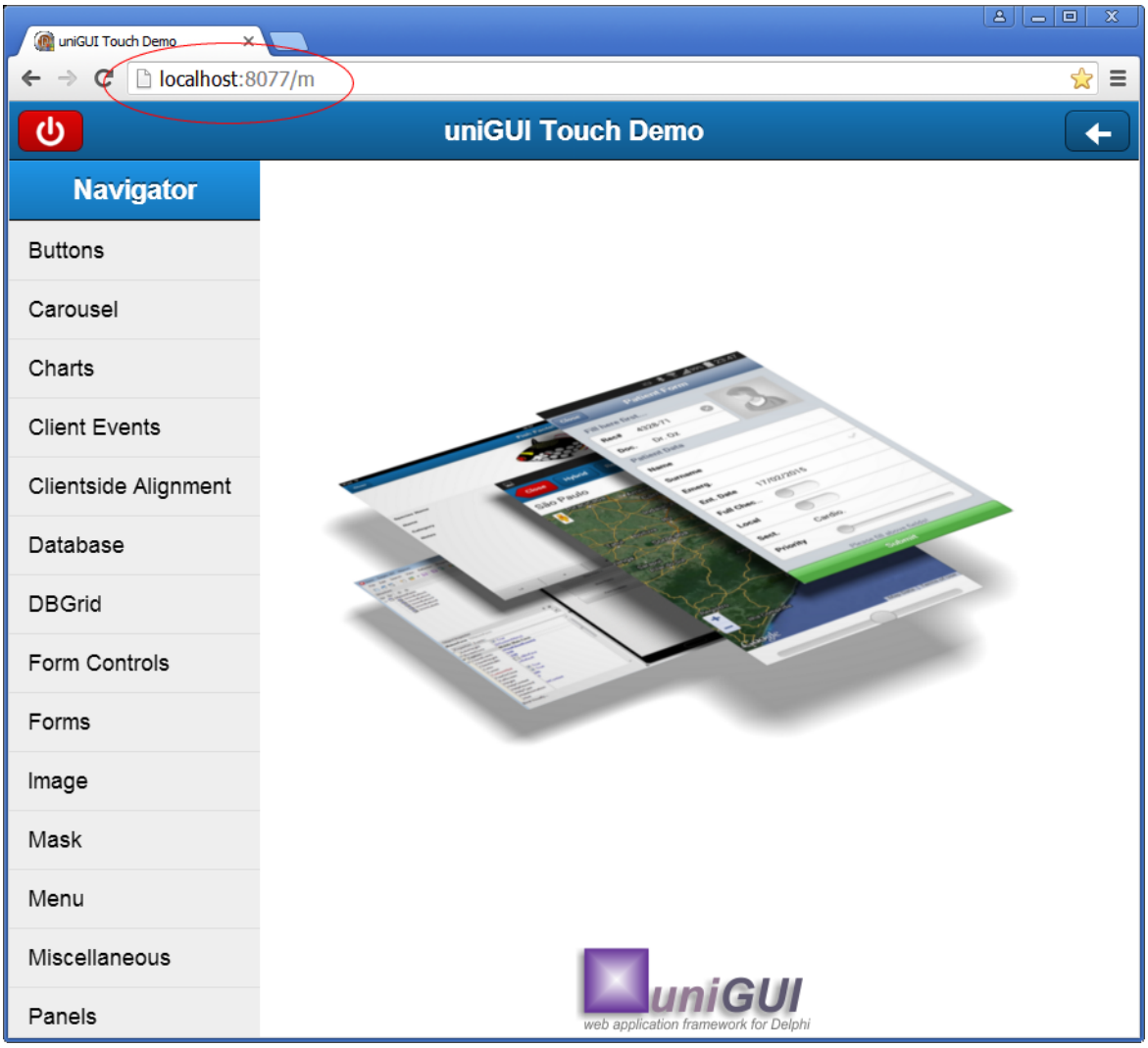
You can test or debug a mobile web session in an ordinary desktop browser or directly run it on a mobile device. On desktop you need a fully HTML5 compatible browser. Fortunately all modern browsers come with extensive support for HTML5.

In Browser address bar type below address:

http://localhost:8077

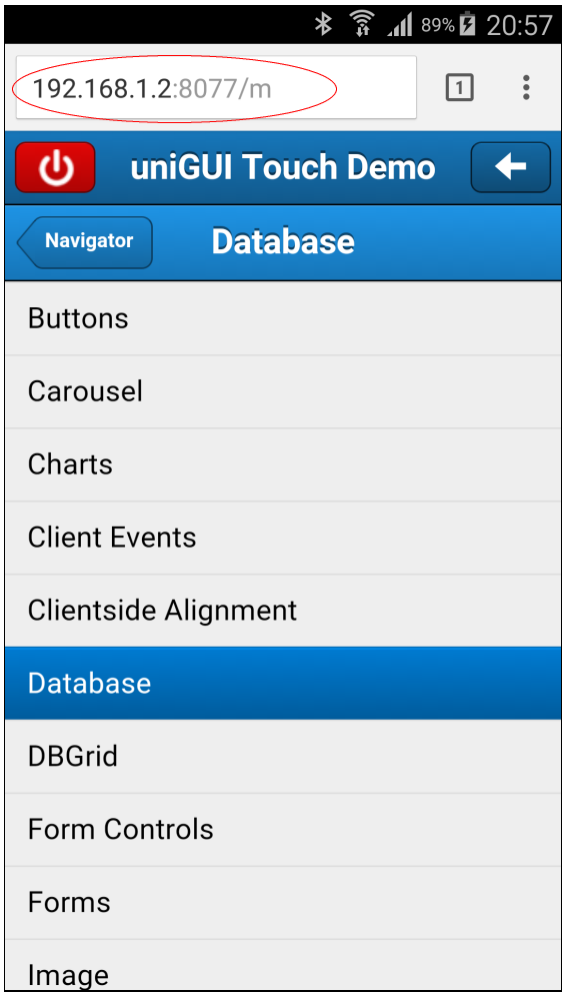
You will notice that address will be automatically redirected to a new address:

http://localhost:8077/m



uniGUI mobile app in Google Chrome browser

You can also run on a mobile device by simply opening a browser window and type a proper address. This time you need to enter IP address of PC which is running demos and also make sure your mobile device and PC are connected to same LAN.



uniGUI web application running on a smartphone

3 Technology Overview

Developing a Rich Internet Application (RIA) is one of the challenging fields in software development. Tasks such as handling user sessions, tracking view state, updating web controls, handling AJAX calls and etc. can be very challenging if your development tool doesn't automatically handle all of these for you. uniGUI Web application framework makes developing stateful Web applications easier than ever. uniGUI extends Web Application development experience to a new dimension. Each uniGUI application can be considered both as a standard Delphi VCL application which uses web as its presentation layer. uniGUI enables developers to create, design and debug their Delphi applications as if they are developing regular desktop applications and then choose one of the available options for web deployment. uniGUI itself is not a single library. For Web front-end uniGUI relies on well known Sencha Ext JS JavaScript library. Thanks to Ext JS for enabling uniGUI to provide a high end, visually perfect and fully AJAX enabled web front-end.

uniGUI uses same Form oriented application model available in a typical VCL application. This means your application will be consisted of many forms which all belong to same session. This will enable developers to follow same principles and design patterns they were following in designing standard desktop application. This will highly reduces learning curve which can be very steep for other web application frameworks. Especially for those which need developers' direct interaction with UI design details, CSS code, HTML code, XML templates and JavaScript code. This means that a Delphi developer with little web knowledge can start developing web application using uniGUI out of the box. While uniGUI enables developers to develop web applications with little knowledge for web technologies, it is always recommended for developers to fully become familiar web underlying web technologies which are used in general in the web and used in particular in uniGUI, such Ext JS framework. Becoming familiar with Ext JS framework will open door to a new world which developers can customize their web application in a war which wouldn't be possible otherwise. uniGUI allows developers to directly write JavaScript client side event handlers for Ext JS controls. This advanced feature allows developers to directly allow interaction between client side screen elements without a need to communicate with the server.

In general each uniGUI application is a standard Delphi executable powered with specialized modules which all together turn the application into a full featured web server. Among these modules we can count Session Manager, Web HTTP Server, ISAPI Handler, Cache Eraser and Server Module. These modules cooperate to create sessions, handle Ajax calls, create web UI and manage session lifetime. All these actions happens completely transparent to developers.

3.1 Unified GUI

uniGUI stands for **Uni**fied **G**raphical **U**ser **I**nterface or **Uni**fied **GUI** in short. It is called unified because it allows same **UI** experience in all devices with a web browser. Regardless of device, OS, CPU and display same level of user experience can be achieved on all devices with a compatible web browser. It allows a great freedom in choosing client devices. Client device can be anything from a Windows PC, OSX device to a PC with any flavor of Linux or even a Raspberry Pi!

Of course, this feature is not something unique to uniGUI. This level of independency from platform is something that any web application can provide except that uniGUI enables you to create web applications which are very close to desktop applications in look and feel.

MacBook

Windows PC

ChromeBook

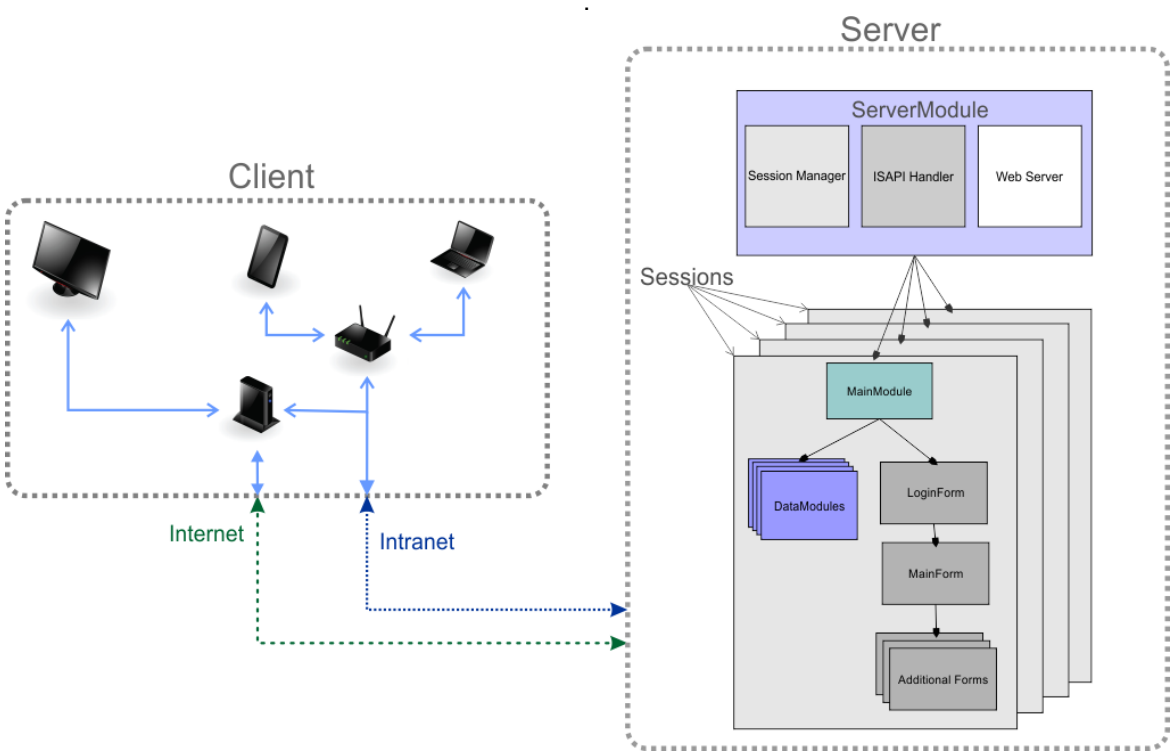
Mobile Devices

Ubuntu Notebook

3.2 Web Sessions

Web Sessions are main elements of a uniGUI web application. Each time a user opens a new instance of web application a new session on server is created. Each session will remain active in server until user terminates it or a timeout condition occurs. Each session keeps a complete state of running web application. That's why uniGUI sessions are called stateful. You can consider each session as a private copy of your web application which co-exists with other sessions in server's address space. Each session is isolated from other sessions and each web request is automatically redirected to its respected session. Each session has a unique "Session Id" which is used to distinguish it from other sessions. Session Id is assigned upon creation of session and included in each Ajax request, so request can be directed to correct session.

Below diagram represents internal structure of a uniGUI server. Each uniGUI server has a single copy of *ServerModule* which is created once per server, along with multiple sessions which are dynamically created and destroyed according to user activity. A uniGUI session contains a specialized *DataModule* called *MainModule* which is automatically created for each session. Also a Form named *MainForm* is created which apparently is main entry point of web application. *MainForm* can be preceded with a *LoginForm* which provide a reliable and secure way for user login. As expected each session can contain several additional *DataModule*s and Forms.



3.3 Deployment Options

uniGUI supports all major deployment options available for Windows platform. Basic method of deployment is Standalone Server. In this mode application server runs directly like a desktop application. It is also the mode used for debugging application. After running application executable it will minimize itself to tray icon and will run until manually terminated by user. See below pic:

uniGUI Standalone Server

A standalone application can be accessed from browser by simply typing:

http://localhost:8077

Where 8077 is dedicated port number which your application is bound. It can easily be modified from ServerModule.

Standalone mode is only recommended for debugging purpose. Since it runs as a desktop application it will be terminated as soon as current user is logged off. It also will not start automatically after a restart. This mode is best for debugging purpose. When your app runs in debug mode you can set breakpoints, pause, go to cursor and use all other advanced debugging features of Delphi IDE to debug your application just like any other VCL application. Apparently, this method is not recommended for production environment, as it doesn't automatically run when OS restarts. Also it can be easily terminated by an unauthorized user intervention.

Another deployment method is Windows Service. By creating a uniGUI Windows Service application you can deploy your application as a standard Windows Service application. This method is one of the preferred method for use in a production environment. Windows Services are integrated part of OS and they run automatically each time system restarts. This will guarantee required availability expected from a web application. A Windows Service application can be accessed from a browser with same method as Standalone server described above. Which means each Windows Service application requires a dedicated port.

Last available deployment option is ISAPI Module. This method is introduced in Microsoft IIS server. It is available from early versions of IIS and it is based on Windows DLL technology.There are other web servers such Apache which also support loading ISAPI modules. ISAPI modules differs from other two options which discussed above in many aspects. First of all it doesn't contain with a built-in web server as opposed to Standalone Service and Windows Service which both embed a web server to serve incoming HTTP requests. In ISAPI mode IIS server is the HTTP server and ISAPI module performs as an application only. You can easily create ISAPI module application using uniGUI Wizards in Delphi IDE. Compiling an ISAPI uniGUI application will output a DLL file instead of an EXE file.This DLL file must be deployed to IIS server which is described in detail in section ISAPI Module under Web Deployment section. uniGUI DDLs support all IIS versions starting from IIS 5.1 and above. ISAPI modules gives developer the freedom of deploying many modules in same server without a need to choose a different port for each application. It also take benefits from all advanced security features available in Microsoft IIS.

Running an ISAPI application is as easy as calling below Url in your browser:

http://localhost/appdir/app.dll

if you have multiple apps under same folder you can call them by simply specifying a different DLL name:

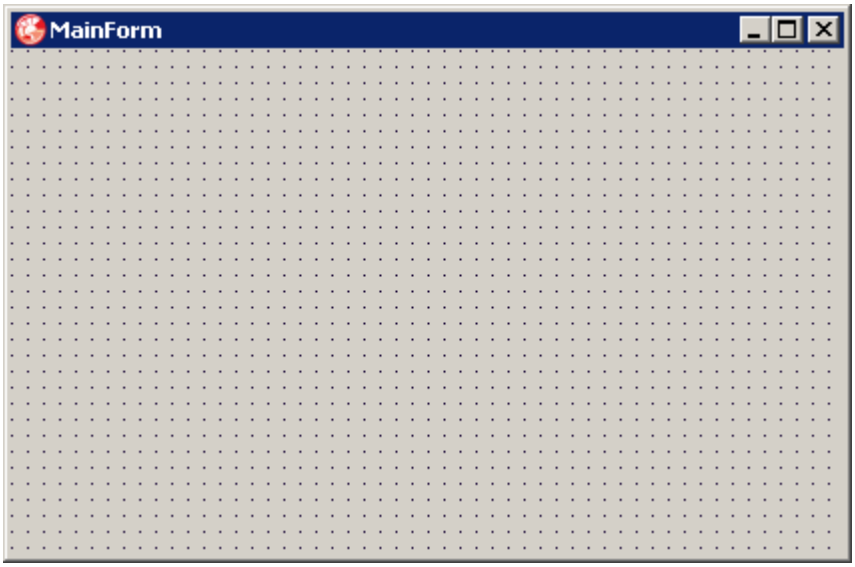
http://localhost/appdir/app.dll
http://localhost/appdir/app2.dll
http://localhost/appdir/appaccount.dll

3.4 Forms and Modules

Unlike a VCL application which can contain only a single form, a uniGUI application is created with a specialized form named MainForm and two data modules MainModule and ServerModule. Other form type which has a special meaning in uniGUI is LoginForm.

3.4.1 Application MainForm

Application Main Form or **MainForm** is the first displayed form when a web session starts. In general MainForm is first form of your application which is used to navigate to other forms using menus or other navigation tools. MainForm is automatically created when a new project is created. Each web session has its own private copy of MainForm and closing a MainForm will terminate the session.



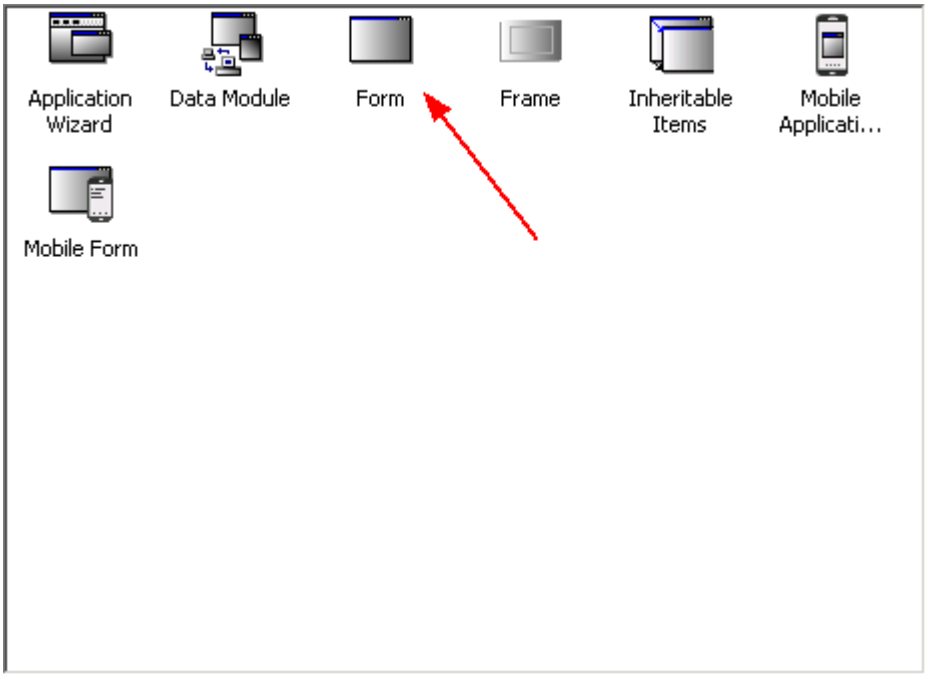
A blank MainForm

MainForm is registered in `initialization` section of unit so uniGUI can distinguish it from other forms.

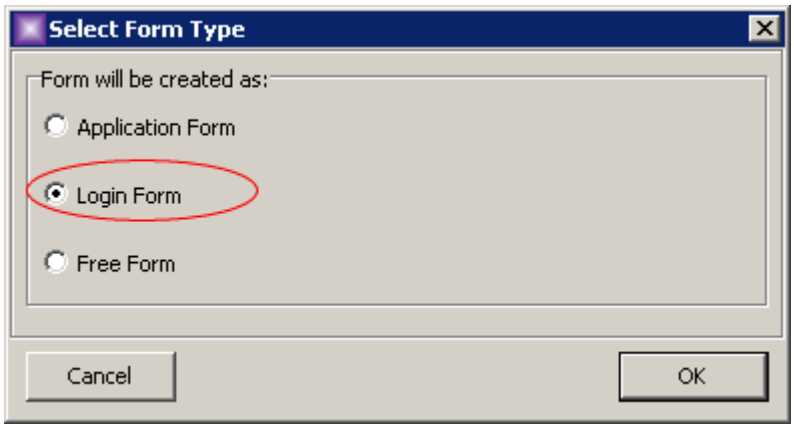
```
initialization
  RegisterAppFormClass(TMainForm);

end.
```

3.4.2 LoginForm

LoginForm is another special Form type which is solely used for login purpose. If your application contains a LoginForm it will the first form displayed when a web session starts. A LoginForm can be created from uniGUI Wizard by following this path: *File->New->Other->Delphi Projects->uniGUI for Delphi->Form*.

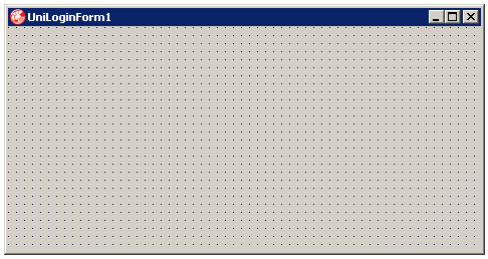


uniGUI Wizard

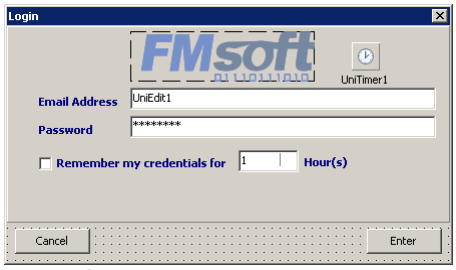


Create a Login Form

This action will create a blank LoginForm which looks identical to a regular form:



A blank LoginForm



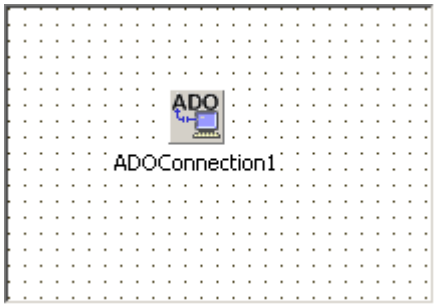
Sample LoginForm design

A LoginForm is a descendant of a built-in class named **TUniLoginForm.** Each application can only has one LoginForm. After adding LoginForm your application will show this form as default form when a new session starts. You need to add controls, event handlers and etc. to implement required functionality. Login behavior is controlled using Form's ModalResult. If LognForm returns *mrOK* it means a successful login which will proceed to MainForm. When ModalResult returns mrCancel it will terminate application. So what you need to do is to validate user input and once there is a valid combination of user name and password you must return *mrOK* as ModalResult.

```
procedure TUniLoginForm1.UniButton1Click(Sender: TObject);
begin
  ModalResult := mrOK;  // Login is valid so proceed to MainForm
end;

procedure TUniLoginForm1.UniButton2Click(Sender: TObject);
begin
  ModalResult := mrCancel; // Invalid Login exit from app
end;
```

Once user is logged in and MainForm is displayed, there are two ways to terminate the session. You can terminate the session and return to LoginForm by returning *mrOK* as ModalResult or terminate the session by returning *mrCancel*. For security reasons existing session is always terminated before displaying the LoginForm. i.e. each new login.starts a new session by default.

3.4.3 MainModule



Application MainModule

MainModule can be considered as heart of each session. It is a special purpose DataModule which is automatically created and added to project each time a new project is created. MainModule has a many important roles in a uniGUI application. Some of these roles are not visible to the developers. For developers MainModule can be a used a central module to place session's shared resources, such as database connections, shared variables and etc. For example, you can declare public variables in MainModule's public section and then access it from other forms in session. Below example demonstrates a common practice in uniGUI to share data among various forms in a session. Since each session has its private copy of MainModule it will be ensured that each form will access its private set of data in its session.

```
  TUniMainModule = class(TUniGUIMainModule)
  private
    { Private declarations }
  public
    { Public declarations }
    aUserName, aPassword: string;
  end;
```

Later you can access these variables from other forms in application:

```
procedure TMainForm.UniButton1Click(Sender: TObject);
begin
  UniLabel1.Caption := UniMainModule.aUserName + ' ' + UniMainModule.aPassword;
end;
```

Also see Web Sessions.

3.4.4 ServerModule

uniGUI ServerModule

Each uniGUI application contains a special data module named ServerModule which is the

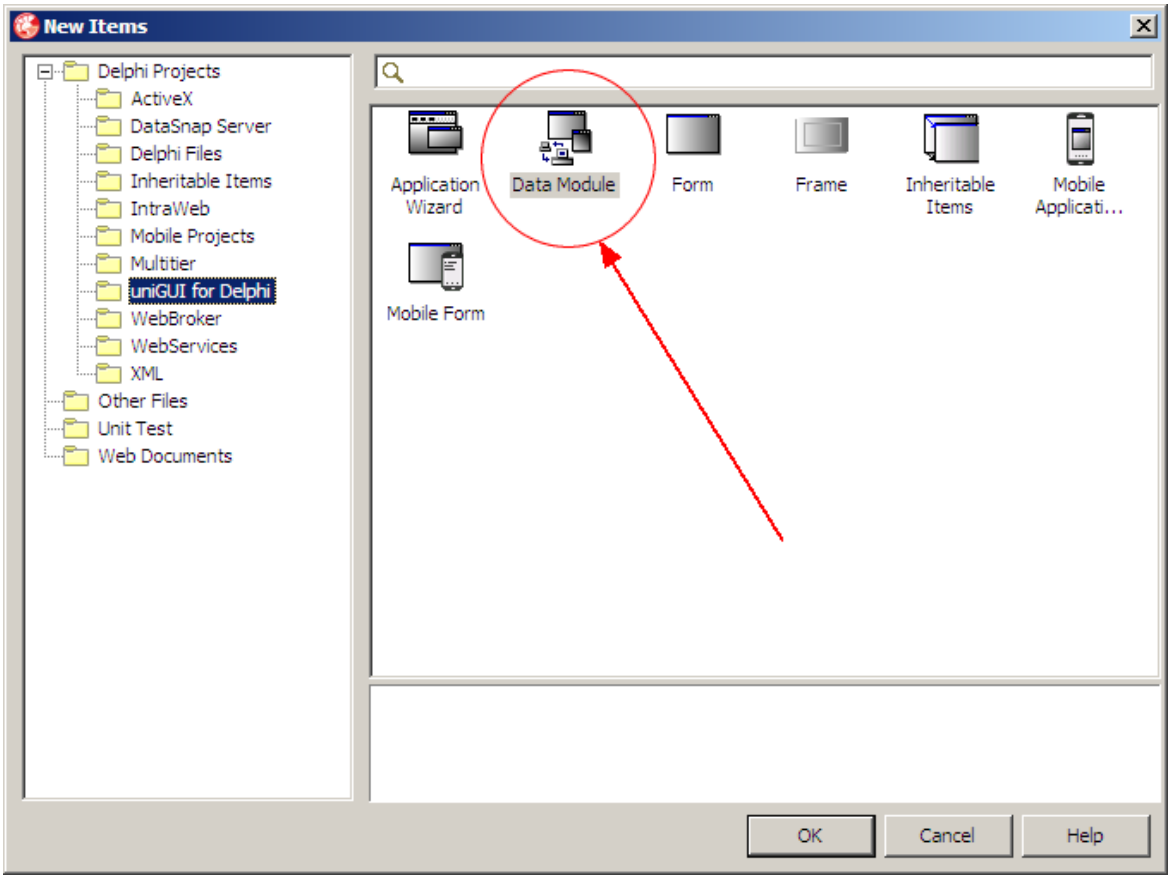
core module of the application. It is a singleton, which means it is created only once per application. It is mainly used to configure various server settings. ServerModule will be covered in more details in other sections.

3.4.5 ServiceModule

A ServiceModule is only created when project type is Windows Service. It a descendant of Delphi's standard TService class. It allows to configure Windows Service related parameters including service name, service type and etc.

3.4.6 DataModules

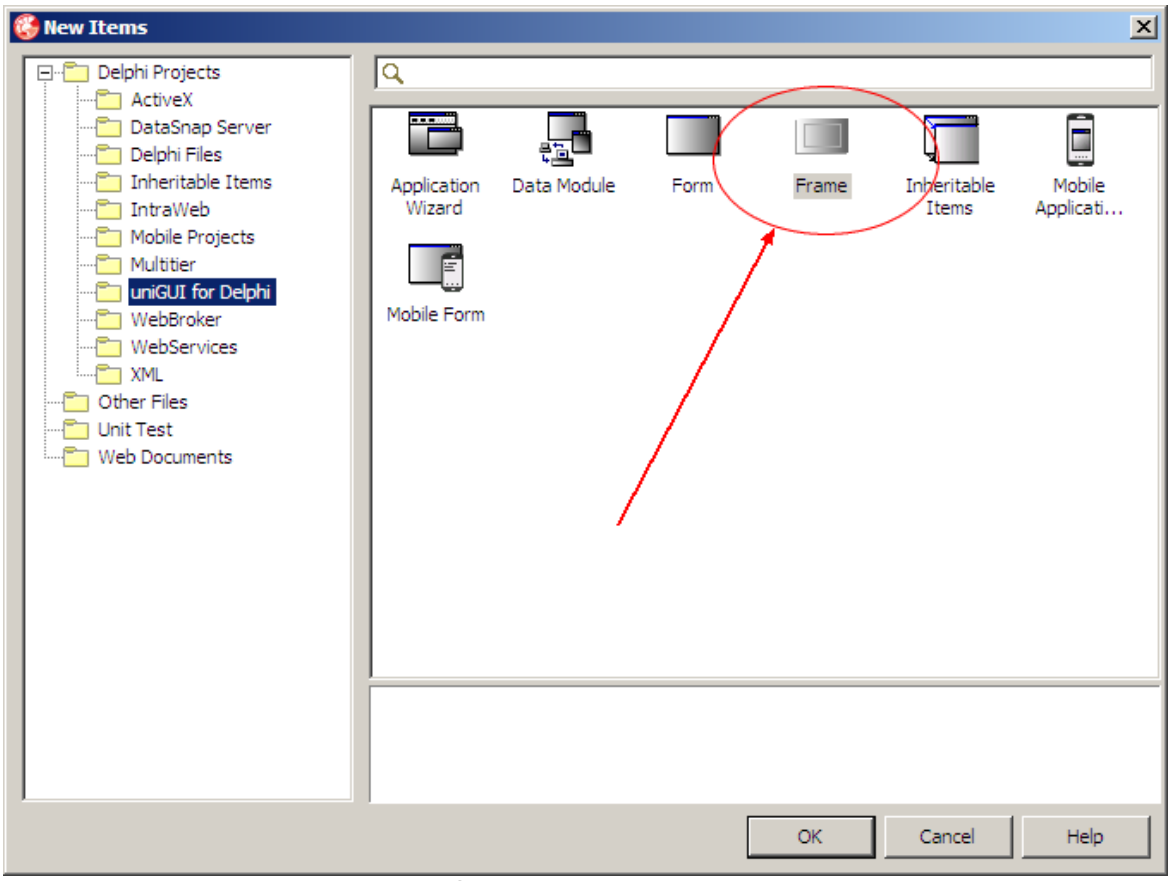
uniGUI supports adding DataModules to project. This will allow developers to design their app like a standard VCL application divide application business logic among several DataModules. The important thing to remember is that while uniGUI DataModules are identical to standard VCL DataModules in nature, they must be created using uniGUI wizard to ensure each session will have its own private copy of that DataModule. If DataModule is created using standard IDE wizard that module will not be compatible with uniGUI session management system so it should be avoided.



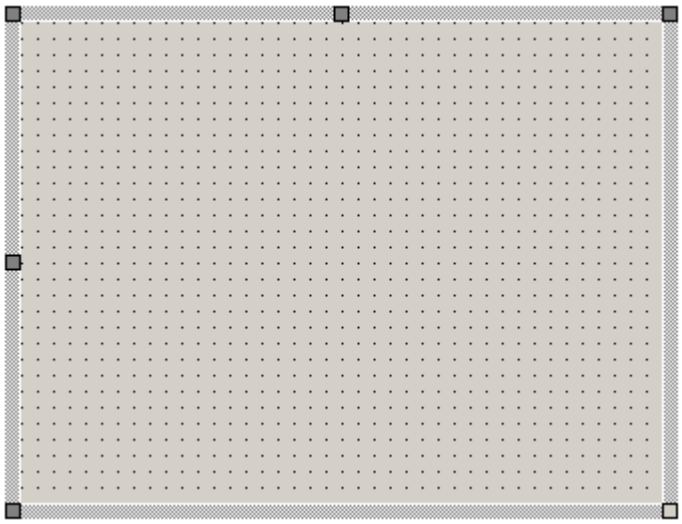
Using uniGUI wizard to create a new DataModule

3.4.7 Frames

uniGUI has complete support for frames. Frames must be created using uniGUI IDE wizard.



Using uniGUI wizard to create a new Frame



An empty uniGUI Frame

3.5 Special Objects

3.5.1 UniApplication Object

UniApplication returns an instance of TUniGUIApplication for current session. For each session UniApplication's main role is to be owner for MainForm, MainModule and all other forms and DataModules which belong to current session. It is globally available and can be accessed inside uniGUI control event handlers. UniApplication has several useful properties which can be used to obtain various information regarding the associated session. For example, url parameters, client screen width, client screen height, device type, information related client PC and cookies.

For each session an instance of UniApplication is created and all forms and data modules in projects will be owned by this instance UniApplication. When a form is created in code its owner must be set as UniApplication too.

```
procedure TMainForm.UniButton2Click(Sender: TObject);
begin
  with TUniForm2.Create(UniApplication) do
    ShowModal();
end;
```

UniApplication itself is actually a function which returns the correct instance of TUniGUIApplication for current session:

```
unit uniGUIApplication;

interface

function UniSession: TUniGUISession;
function UniApplication: TUniGUIApplication;
...
```

For this reason UniApplication should only be accessed inside an event handler which is fired from a uniGUI control.

```
procedure TMainmForm.UnimButton1Click(Sender: TObject);
begin
  if upAndroid in UniApplication.UniPlatform then
    ShowMessage('This is an Android device!');
end;
```

3.5.2 UniSession Object

UniSession returns an instance of TUniGUISession class for current session. It contains all information related to a session. Among these there are important information such as IP Address, User Agent, Host Address, Platform related data and etc. UniSession also contains important methods which can be used to control a session. For example you can call Terminate method to terminate a session:

```
procedure TMainForm.UniButton2Click(Sender: TObject);
begin
  UniSession.Terminate;  // Terminate current session

end;

procedure TMainForm.UniButton2Click(Sender: TObject);
begin
  UniSession.UrlRedirect('http://www.newsite.com');  // Redirect current window to a new locati

end;
```

Like UniApplication, UniSession object has a valid instance only when accessed from a uniGUI control event handler. Here we are referring to controls which belong to a session. For example, a TUniButton instance always belongs to a session.

```
procedure TMainForm.UniButton2Click(Sender: TObject);
var
  IPAddress : string;
begin
  IPAddress := UniSession.RemoteIP;  // We are in an event handler from a TUniButton, so UniSes

end;
```

In below example accessing UniSession object will cause an access violation because UniThreadTimer is not an uniGUI control and its events are not associated with any session. UniThreadTimer events run asynchronously in a separate thread.

```
procedure TMainForm.UniThreadTimer1Timer(Sender: TObject);
var
  IPAddress : string;
begin
  IPAddress := UniSession.RemoteIP;  // this will cause an access violation error because there

end;
```

It can be confusing that how UniSession can be a global object and still hold different values when accessed from different sessions. The answer is that just like UniApplication, UniSession is actually a global function declared in *uniGUIApplication.pas.* This function returns correct session instance when called inside an event handler.

```
unit uniGUIApplication;

interface

function UniSession: TUniGUISession;
function UniApplication: TUniGUIApplication;
...
```

3.5.3 UniServerInstance Object

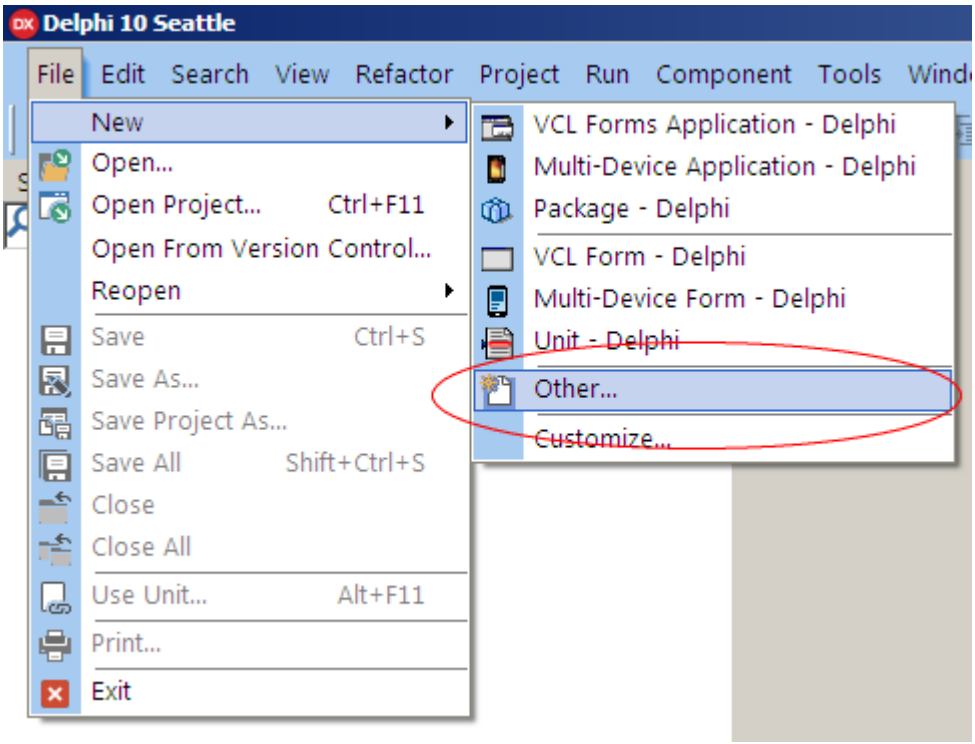
UniServerInstance is a global function defined in uniGUIServer.pas and returns the global instance of ServerModule object which is a singleton. There are some reasons that we have used a function instead of a variable here. One reason is to make sure that UniServerInstance is accessed as read only.

```
unit uniGUIServer;

interface

function UniServerInstance: TUniGUIServerModule;
...
```

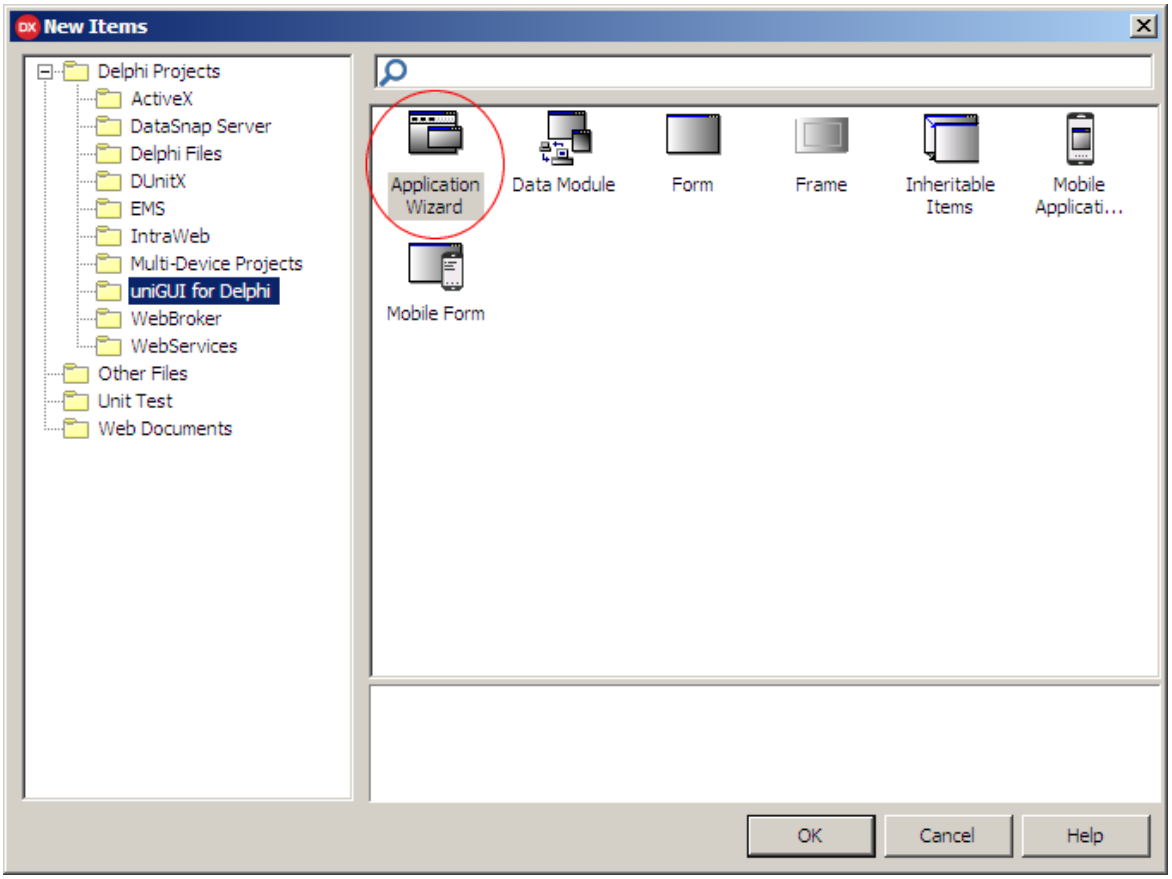
4 Developer's Guide

4.1 Creating a New uniGUI Application

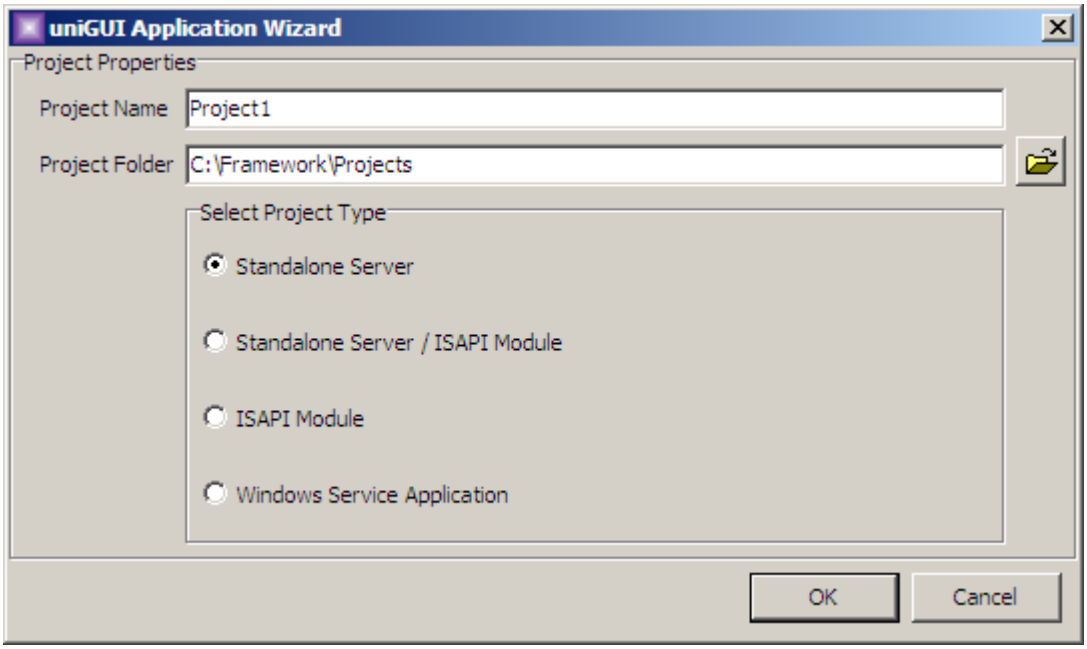
Creating a new uniGUI project is quite easy. All needed is to point Delphi IDE to uniGUI Application Wizard.



Select File->New->Other



Select uniGUI for Delphi -> Application Wizard



Select a Project Type

4.1.1 Standalone Server Project

A Standalone Server project is the most simple type of a uniGUI project. It creates an application which runs similar to a desktop application. i.e.application is started and terminated by user. This type of application is best for debugging purpose. Below is the sample DPR file for a typical newly created Standalone Server project. Also see Deployment Options.

```
program Project1;

uses
  Forms,
  ServerModule in 'ServerModule.pas' {UniServerModule: TUniGUIServerModule},
  MainModule in 'MainModule.pas' {UniMainModule: TUniGUIMainModule},
  Main in 'Main.pas' {MainForm: TUniForm};

{$R *.res}

begin
  Application.Initialize;
  TUniServerModule.Create(Application);
  Application.Run;
end.
```

4.1.2 Standalone Server / ISAPI Module Project

This type of project is a combo project which has a dual type. It can both be a Standalone server and an ISAPI Module. This type of project is very good to start with if ISAPI Module deployment will be targeted for use production. You will be able to convert your Standalone server to an ISAPI DLL by simply commenting out the first line of DPR file. Also see Deployment Options.

```
{$define UNIGUI_VCL} // Comment out this line to turn this project into an ISAPI module'

{$ifndef UNIGUI_VCL}
library
{$else}
program
{$endif}
  Project1;

uses
  uniGUIISAPI,
  Forms,
  ServerModule in 'ServerModule.pas' {UniServerModule: TUniGUIServerModule},
  MainModule in 'MainModule.pas' {UniMainModule: TUniGUIMainModule},
  Main in 'Main.pas' {MainForm: TUniForm};

{$R *.res}

{$ifndef UNIGUI_VCL}
exports
  GetExtensionVersion,
  HttpExtensionProc,
  TerminateExtension;
{$endif}

begin
{$ifdef UNIGUI_VCL}
  Application.Initialize;
  TUniServerModule.Create(Application);
  Application.Run;
{$endif}
end.
```

By default above project will produce an EXE file which is an Standalone server. If you change the first line of project it will turn into an ISAPI module. Later you can convert it back to Standalone EXE mode by removing the comment from first line. Here is how it works.

Steps to convert a combo project to an ISAPI DLL:

1. Open DPR file.
2. Change first line of DPR file to: `{.$define UNIGUI_VCL}`
3. If your Delphi edition is XE2 or newer then you need to close your project and open it again. (This step is not needed if your Delphi version is older than XE2)
4. Build your application.
5. DLL file will be created in related folder.

Steps to convert a combo project back to Standalone mode:

1. Open DPR file.
2. Change first line of DPR file to: `{$define UNIGUI_VCL}`
3. If your Delphi edition is XE2 or newer then you need to close your project and open it again. (This step is not needed if your Delphi version is older than XE2)
4. Build your application.
5. Your project will be compiled to EXE file again.

4.1.3 ISAPI Module Project

If you plan to start an ISAPI Module project and choose it as the default deployment method you can select this type of project. It is apparent from the DPR file that this project will produce a DLL when it is compiled: For deployment options plesee see ISAPI Module section. ISAPI modules can be deployed to Microsoft® IIS server or Apache server for Windows. Also see Deployment Options.

```
library Project1;

uses
  uniGUIISAPI,
  ServerModule in 'ServerModule.pas' {UniServerModule: TUniGUIServerModule};

{$R *.res}

exports
  GetExtensionVersion,
  HttpExtensionProc,
  TerminateExtension;

end.
```

4.1.4 Windows Service Project

Another deployment method for production environment is Windows Service. It creates a standard Delphi Windows Service project with few modifications to allow creation of a uniGUI ServiceModule. Service projects allows creation of standard Windows Service executables which can be installed like any regular Windows Service. A Windows Service is automatically started by OS and it is always available as long as server OS is running. This provides a high level of availability for uniGUI application. Also see Deployment Options.

```
program Project1;

uses
  SvcMgr,
  ServerModule in 'ServerModule.pas' {UniServerModule: TUniGUIServerModule},
  MainModule in 'MainModule.pas' {UniMainModule: TUniGUIMainModule},
  Main in 'Main.pas' {MainForm: TUniForm},
  ServiceModule in 'ServiceModule.pas' {UniServiceModule: TUniGUIService};

{$R *.res}

begin

  if not Application.DelayInitialize or Application.Installing then
    Application.Initialize;
  Application.CreateForm(TUniServiceModule, UniServiceModule);
  Application.Run;
end.
```

4.2 Application Design Considerations

There are certain aspects which make a uniGUI web application different from a desktop application. A desktop applications is designed to run as a single instance on desktop and it is able to use all system resources available to it. Most of the time this will result in poor application design, as developers don't see a need to optimize resource usage. With advancement of computer technology new systems are equipped with more resources with each new model. 10 years ago PCs with 128mb of RAM where considered as entry level while today this limit is upgraded to at least 2 GB of memory. This may lead developers to adopt wrong practices when designing applications which overuse of system resources. Unlike a desktop application where a single user session runs inside a single process, a web application is where several user sessions run in same process instance. In this case resource management can be a vital task which should be taken seriously.

Another important difference is shared resources. Again in a desktop application all resources are dedicated to same user session while in a uniGUI web application resources are shared among several user sessions. Any mismanagement in resources may affect web application server's health which may lead to server instability. One example for resource mismanagement is memory leaks. In a desktop app memory leaks can be a serious issue if desktop app runs for a very long period of time. SInce desktop apps are mainly used for a few hours per day, mainly working hours, such memory leaks will not lead to serious issues. However, in a web application memory leaks can seriously affect server stability in a short period time.

Web application servers are intended to work in a 24/7 manner, so any resource management related issue can build-up over time and cause server to run out of resources. Such conditions can lead to applictaion server instability and eventually to server crash. When discussing shared resources it is good to mention memory corruption problems. Again, in a desktop application memory corruption issues can remain undetected for a long time because such corruptions may not produce dangerous side-effects, but in a web application memory is allocated and disposed at a very high rate in multiple threads, so any sort of memory corruption can lead to serious server instability and erratic application behavior.

Finally, a uniGUI web application is a heavily multi-threaded environment where multiple sessions can run in different simultaneous threads. As a result components used in uniGUI must be fully multi-thread proof. Components which are not compatible with mult-threaded environment will cause serious issues which again will lead to instability of web server.

4.2.1 General Design Concept

Each uniGUI application is created with a MainForm, a MainModule and a ServerModule. As described before MainModule and MainForm are created in a per session basis while ServerModule is a singleton.

4.2.2 Web Application Scalability

Enter topic text here.

4.3 Web Deployment

Currently there are four options available for deploying your uniGUI project to the Web .

Standalone Server
ISAPI Module
Windows Service

4.3.1 Sencha License Considerations

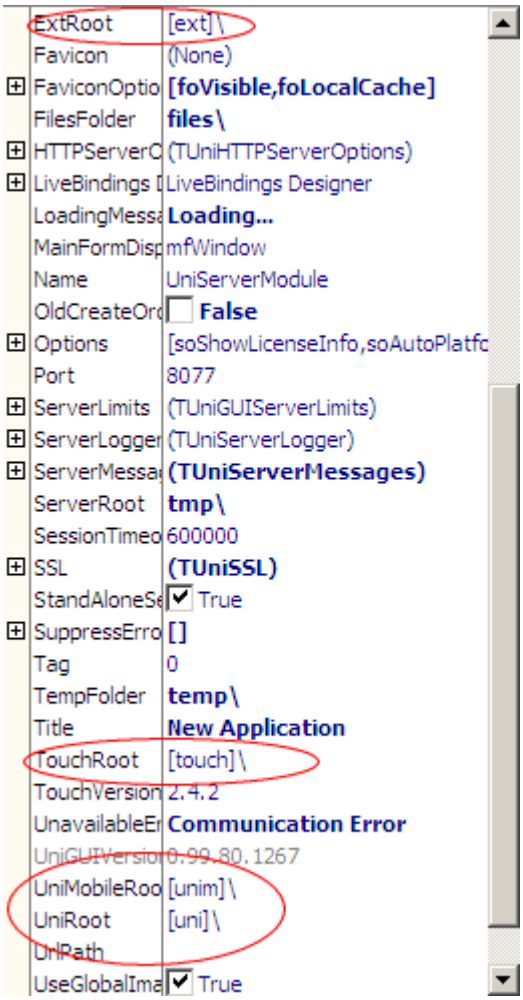
FMSoft Co. Ltd. is an official partner of Sencha Inc. and is granted to distribute **OEM** copies of **Sencha Ext JS** and **Sencha Touch**. **UniGUI** distributes a partial copy of **OEM Sencha Ext JS**. Your **uniGUI** license grants you using and deploying **Sencha Ext JS**. However, your **OEM** copy of **Sencha Ext JS** can only be used in combination with **uniGUI**. You shall not use or distribute **OEM Sencha Ext JS** for any other purpose other than using it to develop and deploy **uniGUI** projects. This library can not be treated as a standalone library and can't be used to create independent software products based on it. **Sencha Ext JS/Sencha Touch** is distributed as an integrated part of **uniGUI** package. **Sencha Ext JS/Sencha Touch** is not distributed in form of a separate product and you are not granted a separate license for it.

4.3.2 uniGUI Runtime Package

uniGUI Runtime Package is an easy way to distribute uniGUI runtime files on server where web applications are deployed. Runtime Package can be downloaded from FMSoft Portal. This package is not available for **uniGUI Trial Edition**. After installing Runtime Package your web application will be able to find related Ext JS and Sencha Touch files required to run your application. **uniGUI Runtime Package** includes a minimal set of **Sencha Ext JS/Touch** library which is required for uniGUI application. It also includes other JavaScript library files that will be needed during runtime.

You must make sure that installed Runtime Package matches version of uniGUI package you have used to compile your web application. For instance, if you have used **FMSoft_uniGUI_Complete_Pro_0.99.80.1267** to build your application, you must also make sure that **FMSoft_uniGUI_Complete_runtime_0.99.80.1267** is installed on your server. You can install multiple versions of runtime package on your server, so web applications compiled with different versions of uniGUI can co-exits on same server.

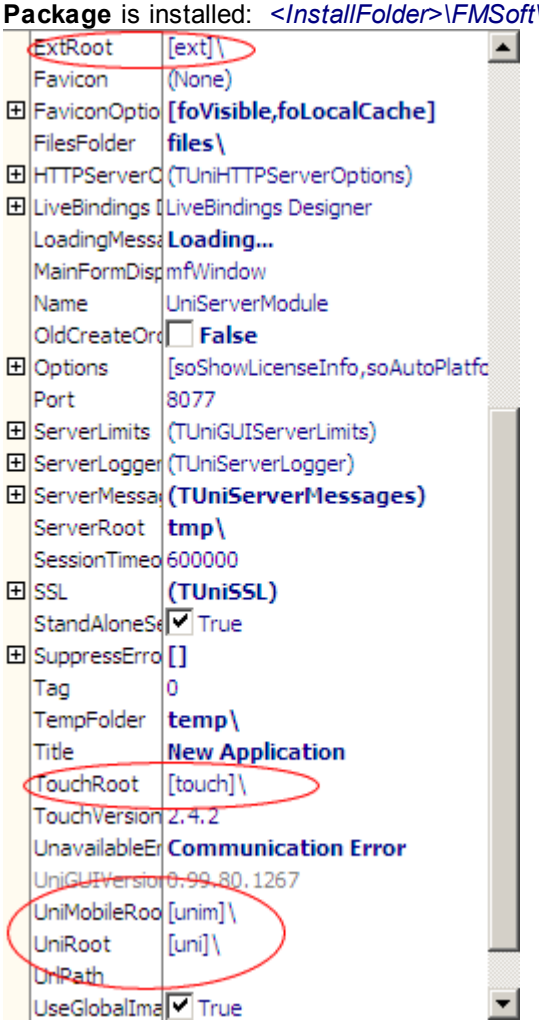
Deploying with Runtime Package requires all related path settings to be their default values. *ExtRoot*, *TouchRoot*, *UniMobileRoot* and *UniRoot* must be set at their default settings. See below:



4.3.3 Adjusting Paths

If you don't plan to deploy your app using uniGUI Runtime Package and you want to deploy runtime files yourself then path settings must be adjusted accordingly. There are some essential paths for a **uniGUI** application which must be adjusted before you deploy runtime files.

First of all, you must be sure that your Web Application knows where **Ext JS** files are located. For this, in your Application ServerModule your must assign a proper path to **ExtRoot** property. Default value of **ExtRoot** is "[ext]\" which means **Ext JS** files are located under where **uniGUI Runtime Package** is installed: *<InstallFolder>\FMSoft\Framework\uniGUI\ext-x.y.z.build*



If you do not install **uniGUI Runtime Package** on target PC, you must assign a full or a relative path to **ExtRoot**. If you assign a relative path it will be relative to **ServerRoot** and you can use the ".. \..\..\myfolder" partial path notation.

The easiest method is to set the **ExtRoot** to **".\[ext]"** and copy the "**ext-x.y.x.build**" folder to root folder of your web application where application executable or dll module resides. However, for security reasons it is better to put "**ext-x.y.x.build**" folder in another folder and deploy all "**ext-x.y.x.build**" files as read-only. Under **IIS** you must be sure that your application has enough credentials for a read-only access to "**ext-x.y.x.build**" folder and its files.

Some Examples:
ExtRoot = "[ext]" ExtJS Files are in *<Runtime Package InstallFolder>\FMSoft \Framework\uniGUI/ext-x.y.z.build\ (*Recommended method)*
ExtRoot = ".\[ext]" ExtJS Files are in *<server root>\ext-x.y.z.build*
ExtRoot = "C:\ExtJS\[ext]" ExtJS Files are in *C:\ExtJS\ext-x.y.z.build*
ExtRoot = ".\ExtJS\[ext]" ExtJS Files are in *<server root>\ExtJS\ext-x.y.z.build*
ExtRoot = ".\ExtJS\ext" ExtJS Files are in *<server root>\ExtJS\ext*

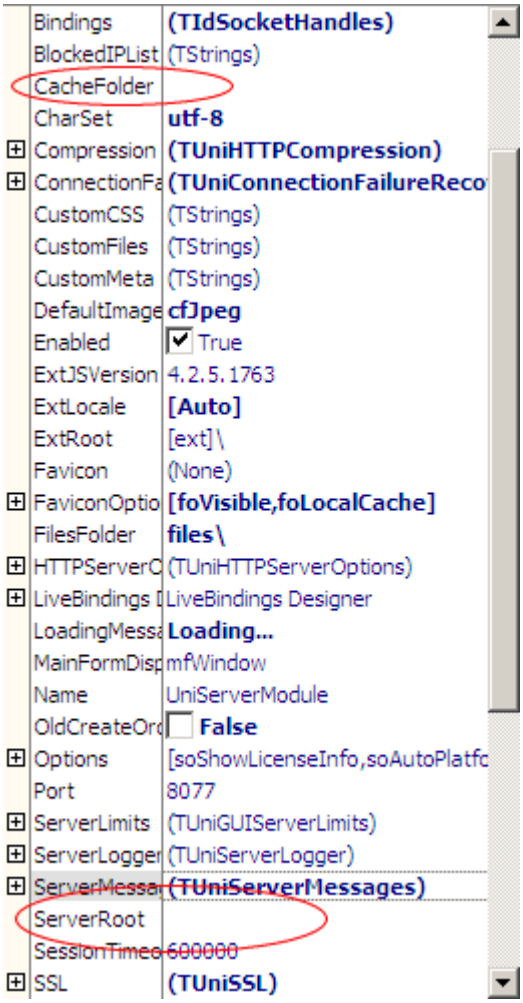
In all cases **ext-x.y.x.build** is translated into a string which represents correct folder for your Ext JS version. For example, if your uniGUI version is based on **Ext JS** version **4.2.5.1736** then it will be translated to folder name: **.\<path>\ext-4.2.5.1763**

Same principle is applicable to *uniRoot*, *uniMobileRoot* and *TouchRoot* properties.

Example:
UniRoot = "[uni]" uniGUI JS Files are in *<Runtime Package InstallFolder>\FMSoft \Framework\uniGUI/uni-x.y.z.build\ (*Recommended method)*
UniRoot = ".\[uni]" uniGUI JS Files are in *<server root>\FMSoft\Framework\uniGUI/uni-x.y.z.build*

In this case **uni-x.y.z.build** is translated into library's **uniGUI** version. If your uniGUI version is **0.99.80.1260** translated folder name will be **uni-0.99.80.1260.**

In **ServerModule** there are two other path related parameters: **ServerRoot** and **CacheFolder**.



ServerRoot:

ServerRoot defines root path for all relative paths. A blank value points to application startup folder.

CacheFolder

A **uniGUI** server needs a folder to create temporary files. Normally, it is a folder named **Cahce** created under same folder your module exists. You can change this by assigning a different path to **CacheFolder** parameter. Under **IIS** you must be sure that your application has enough credentials for a full access to **CacheFolder**.

If you want assign above properties a value at run time the correct location is **OnBeforeInit** event of **UniServerModule**.

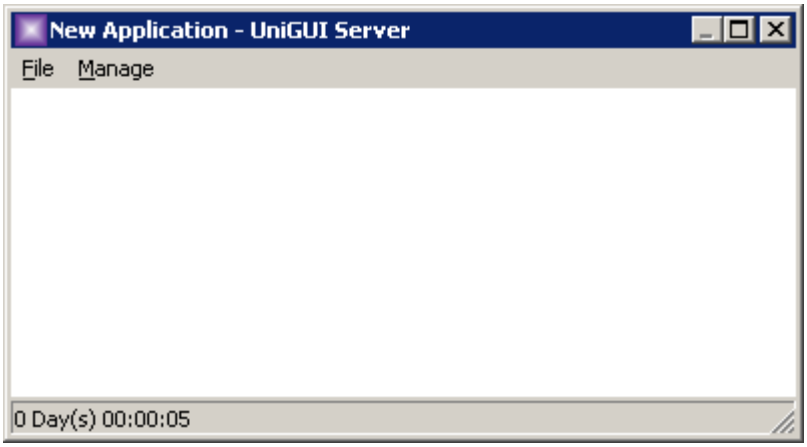
```
procedure TUniServerModule.UniGUIServerModuleBeforeInit(Sender: TObject);
begin
  ExtRoot := 'C:\deploy\[ext]';
  UniRoot := 'C:\deploy\[uni]';
end;
```

4.3.4 Standalone Server

Standalone Server mode is very similar to VCL Application with some differences which makes it a better option for Web deployment. In this mode your application main form is no longer visible on the desktop, instead an icon is placed in Windows taskbar.

uniGUI Standalone Server

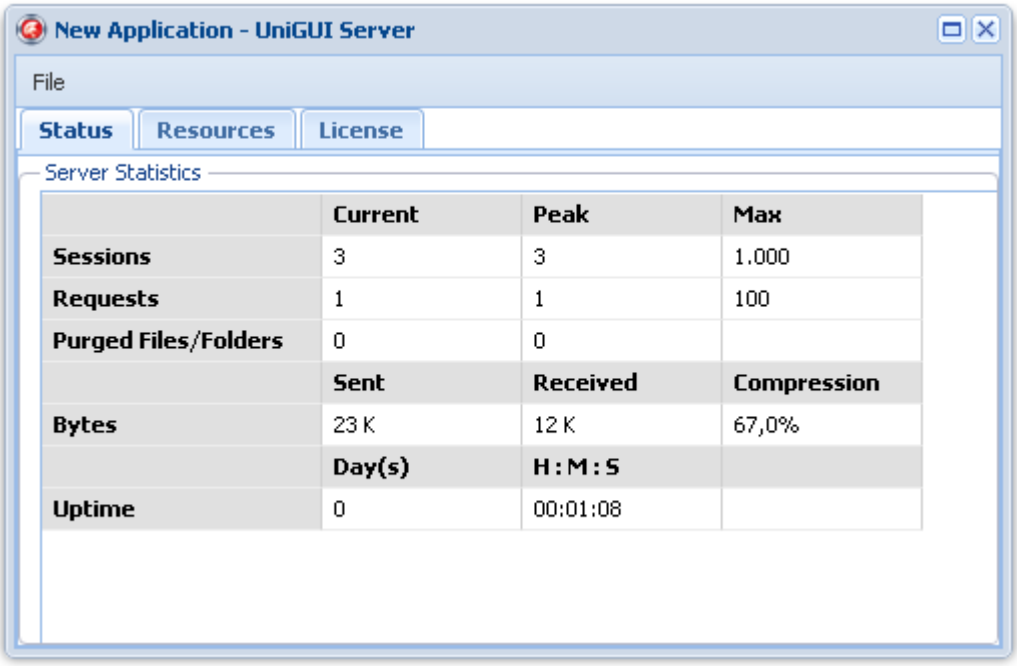
Double-clicking on this icon will open application control panel. Below you see an initial version of control panel.



You can shutdown a running server by right-clicking the icon and selecting Shutdown menu item or you can open server control panel and select Manage->Shutdown from top menu. .

One of the advanced features in uniGUI is accessibility of the Control Panel from web. You can access control panel from this URL:

http://mysite:port/server



Default Icon can be customized by either Changing the Delphi Application Icon or assigning a new Icon to ServerModule->Favicon

Standalone Server is a good option for debugging your application or when you need a web server where server availability is not very important. To automatically start server you must place a shortcut to server executable in Windows **Startup** folder. In this mode server will not start until a Windows user logs in. For serious deployment you must choose either ISAPI Module or Windows Service deployment options.

4.3.5 ISAPI Module

Deploying your Internnet application as ISAPI module probably is best method of deployment. You can run several modules together without a need to dedicate a TCP port to each application.

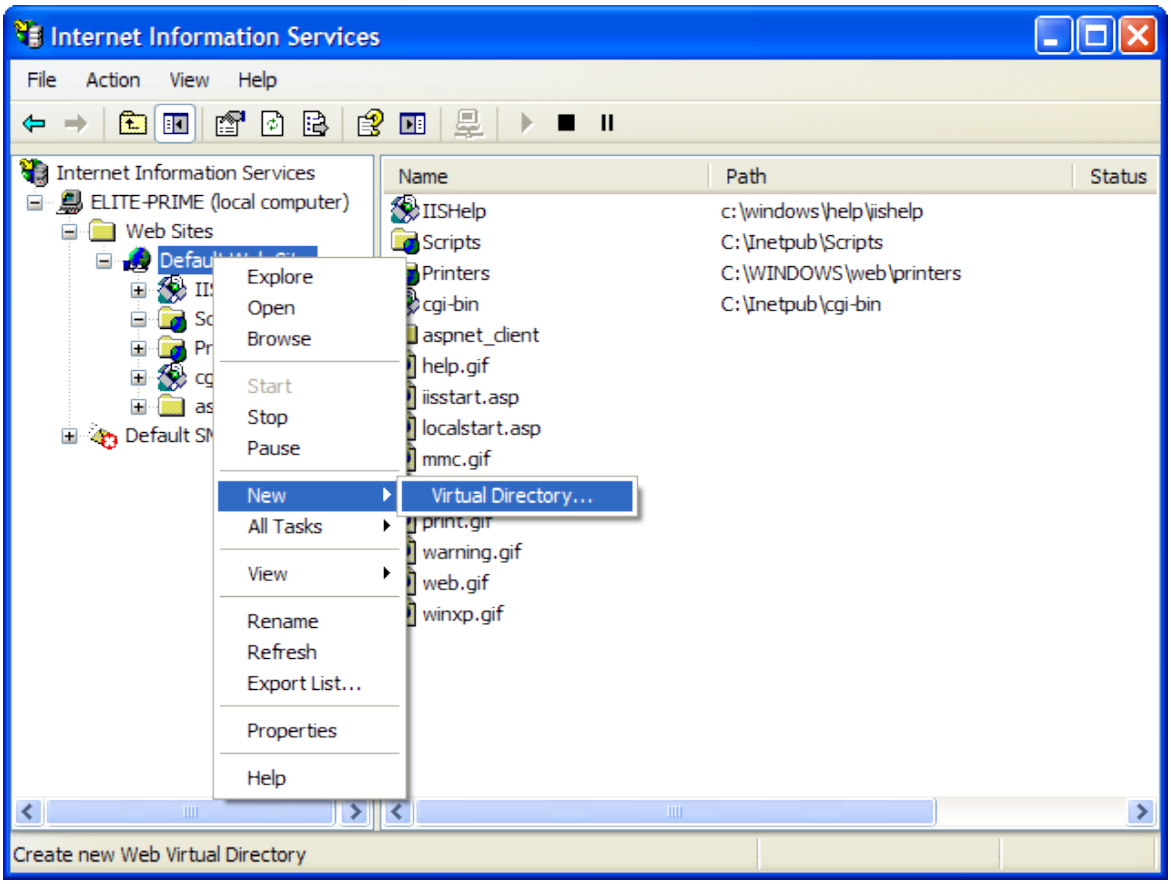
You can use all Web Servers which support ISAPI extentions. uniGUI generated modules are tested with IIS 5.1, IIS 6.0, IIS 7.0 and Apache 2.2.

Installing instructions are different for each Web server. Please refer to below sections for instructions:

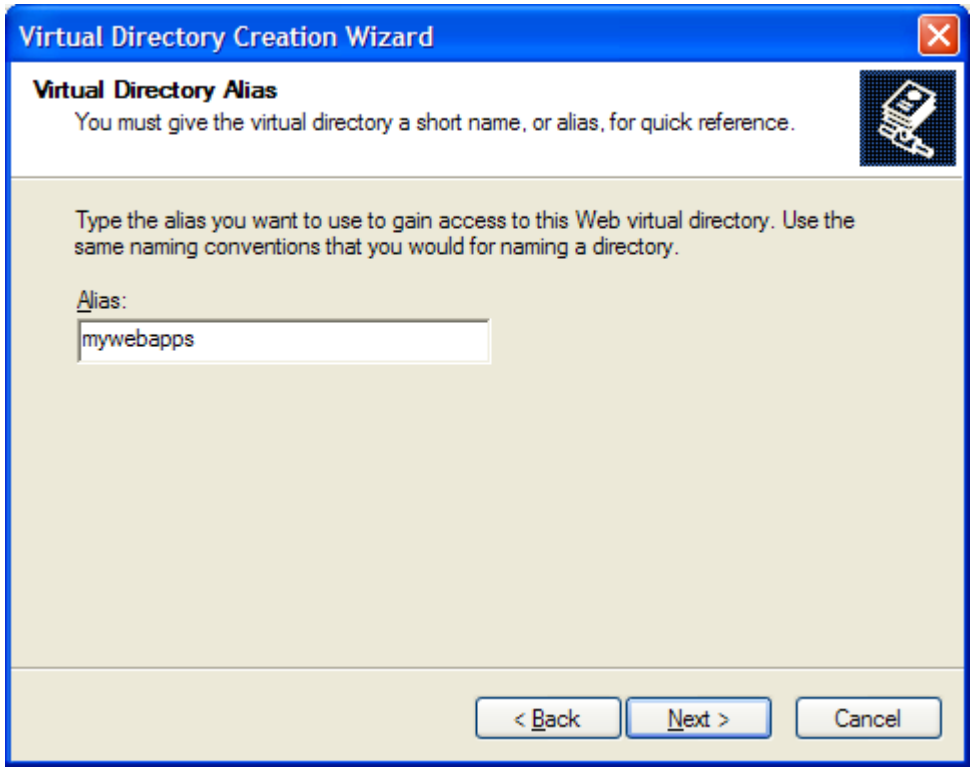
IIS 5.1
IIS 6.0
IIS 7.0
Apache 2.2

4.3.5.1 IIS 5

In order to setup an IIS 5 uniGUI web application first step is to create a new Virtual Directory.

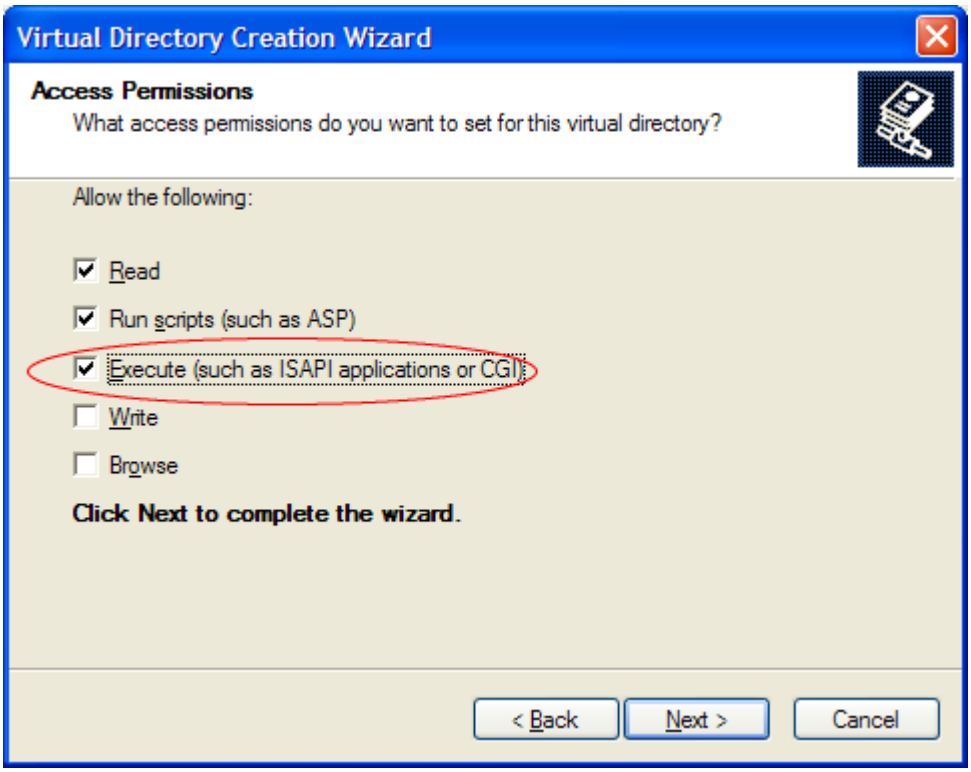


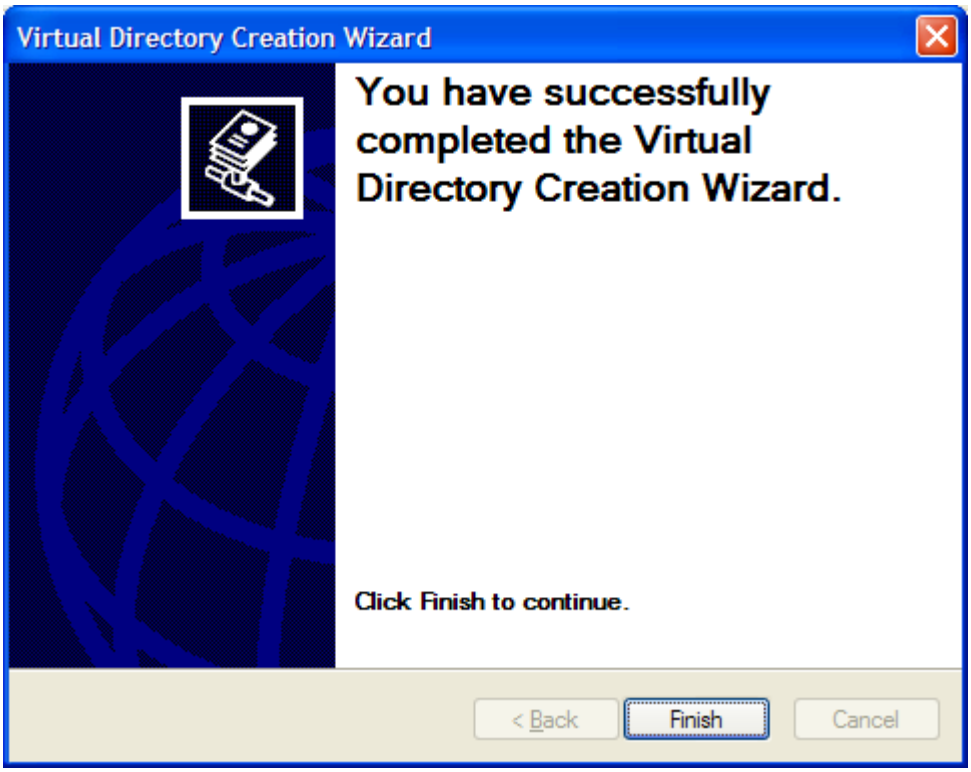
Select an alias for new virtual Directory.



Now assign a folder to newly created alias.

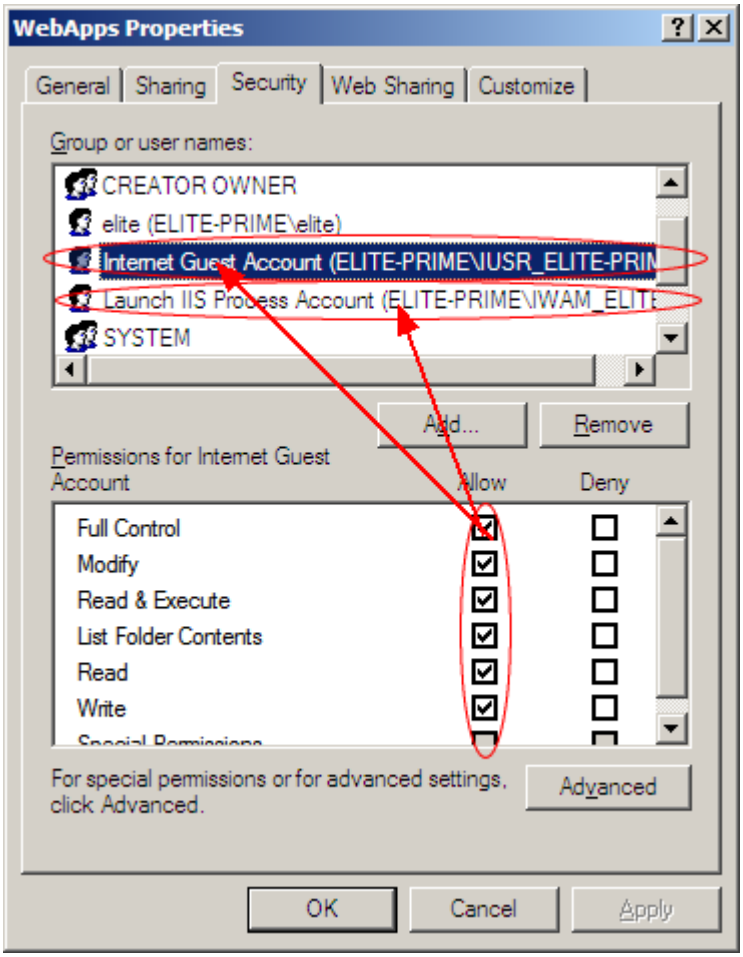
In next step give necessary permissions. **Execute** permission is required.





After creating a new virtual directory you are ready to deploy your uniGUI server.Copy your ISAPI module and other required files to virtual directory. You must be certain that **ISS** built-in users **IUSR_<ComputerName>** and **IWAM_<ComputerName>** has enough credentials to access your virtual directory and other folders that may be accessed during web application execution.

For instance if your ISAPI apps are under folder **C:\WebApps** then you must give full access to

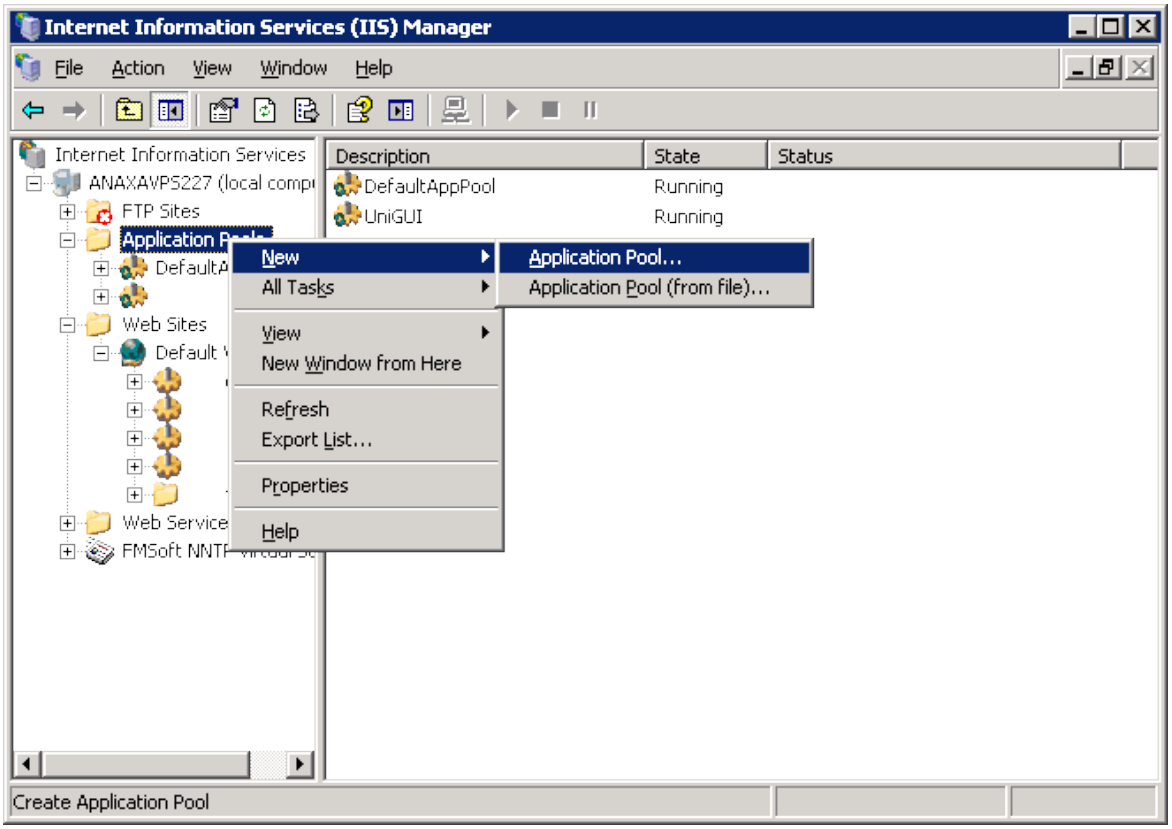


Your web application can be loaded by browsing to this URL: *http://localhost/<virtualdirectory>/<modulename>.dll*

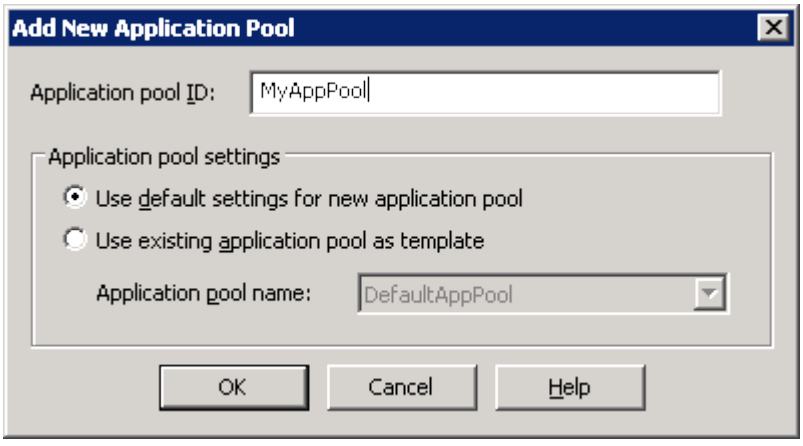
Also refer to Adjusting Paths for more information.

4.3.5.2 IIS 6

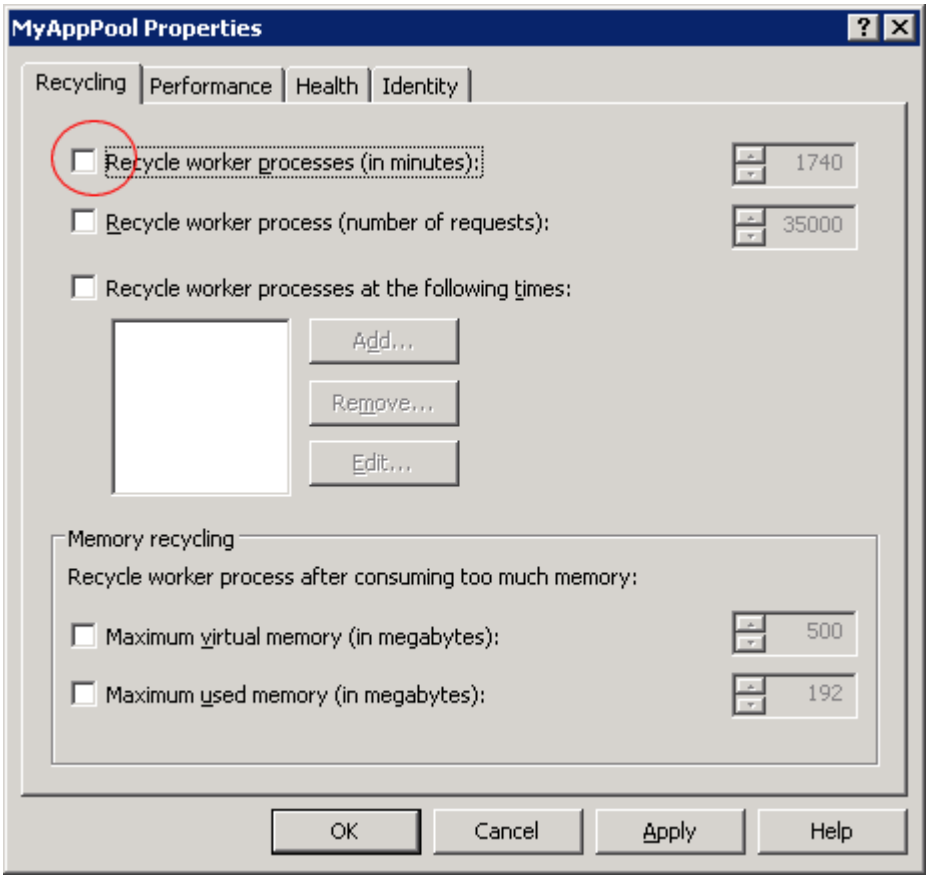
In IIS 6 first step is to create a compatible application pool.



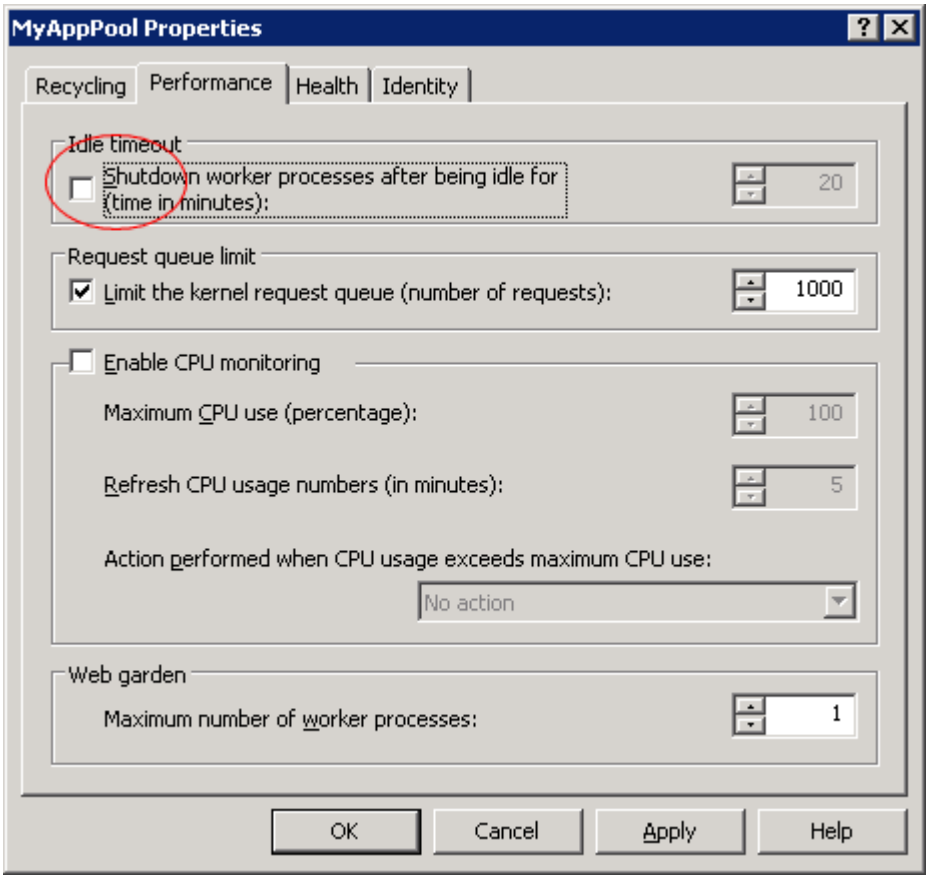
Give a proper name to the new pool.



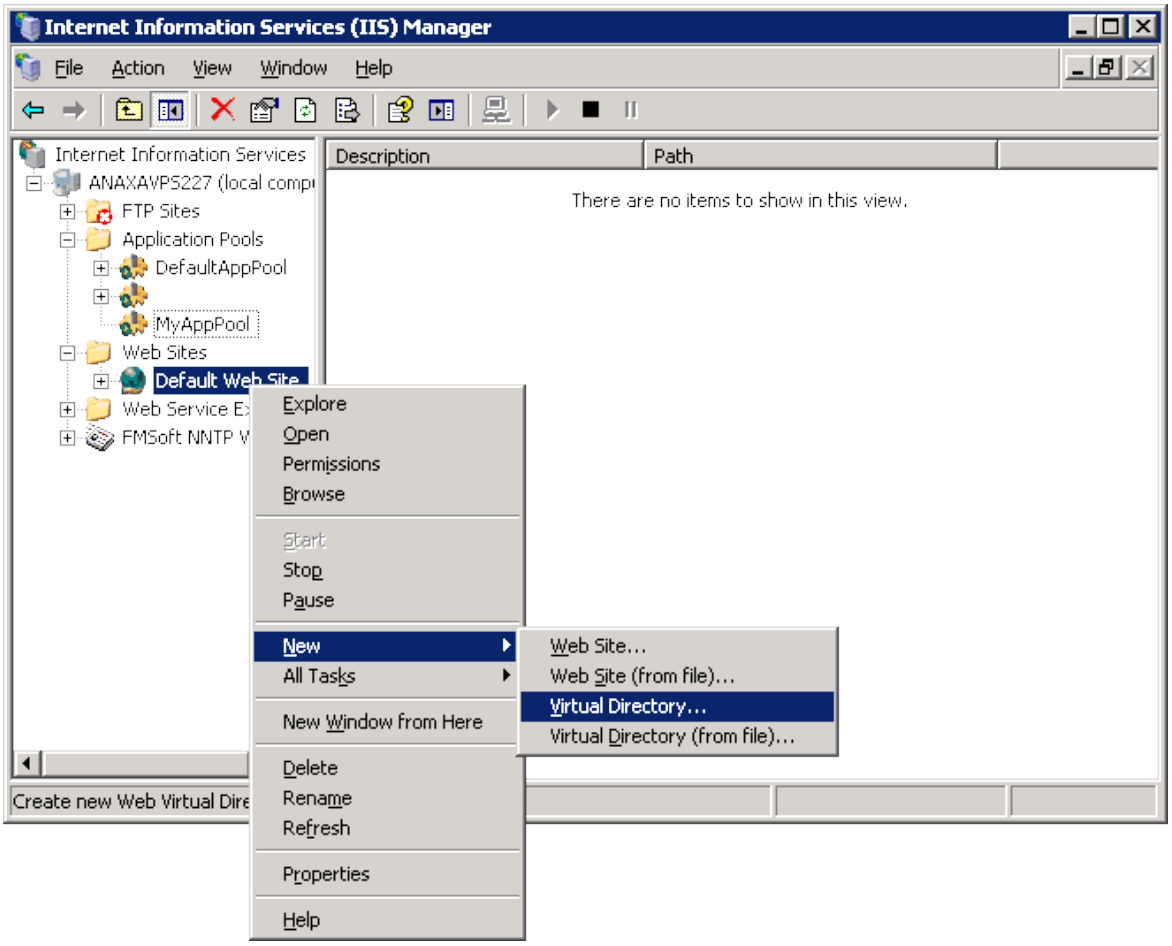
In Application Pool properties, Recycling Tab uncheck the **Recycle worker processes** option.



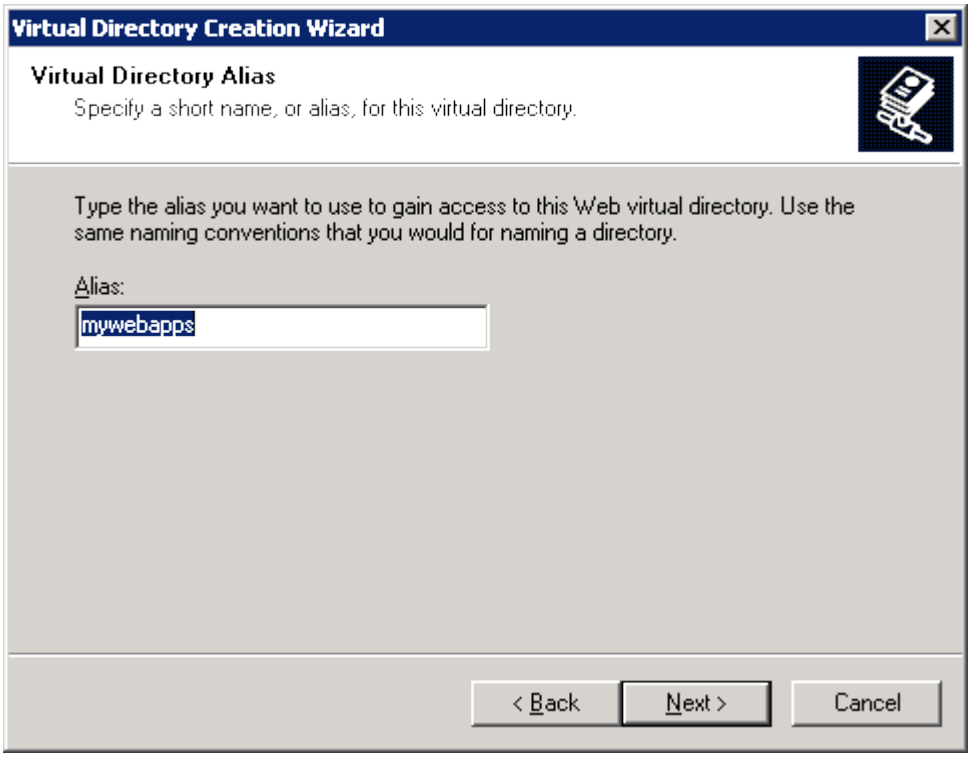
In Performance Tab uncheck the **Shutdown worker processes after being idle...** option.



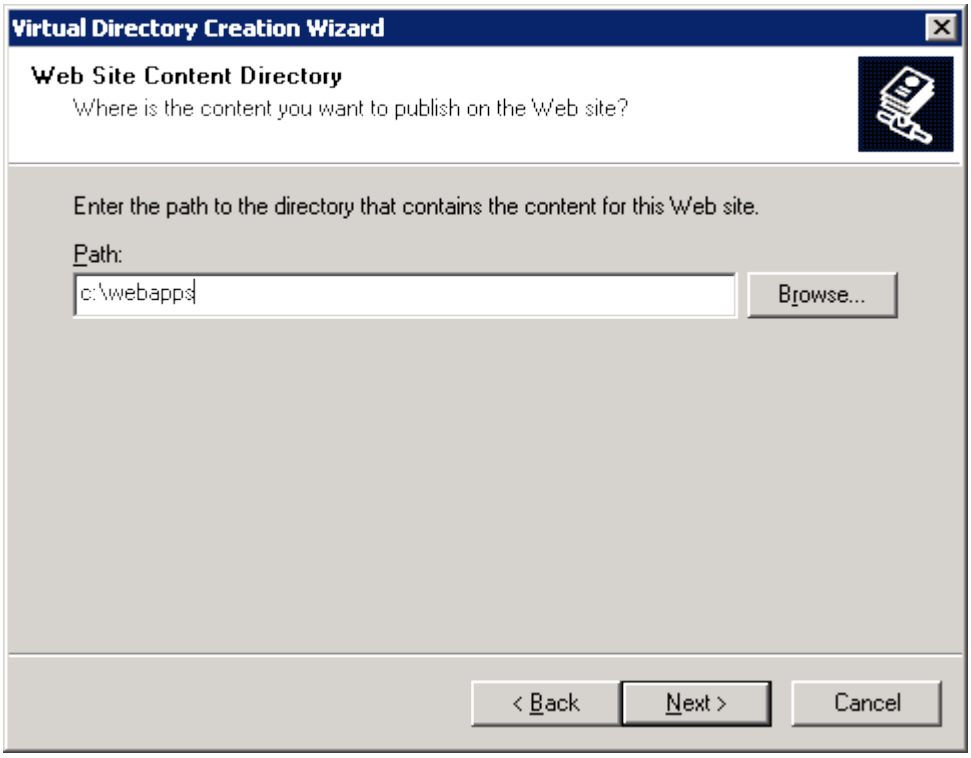
Next step is to create a Virtual Directory.



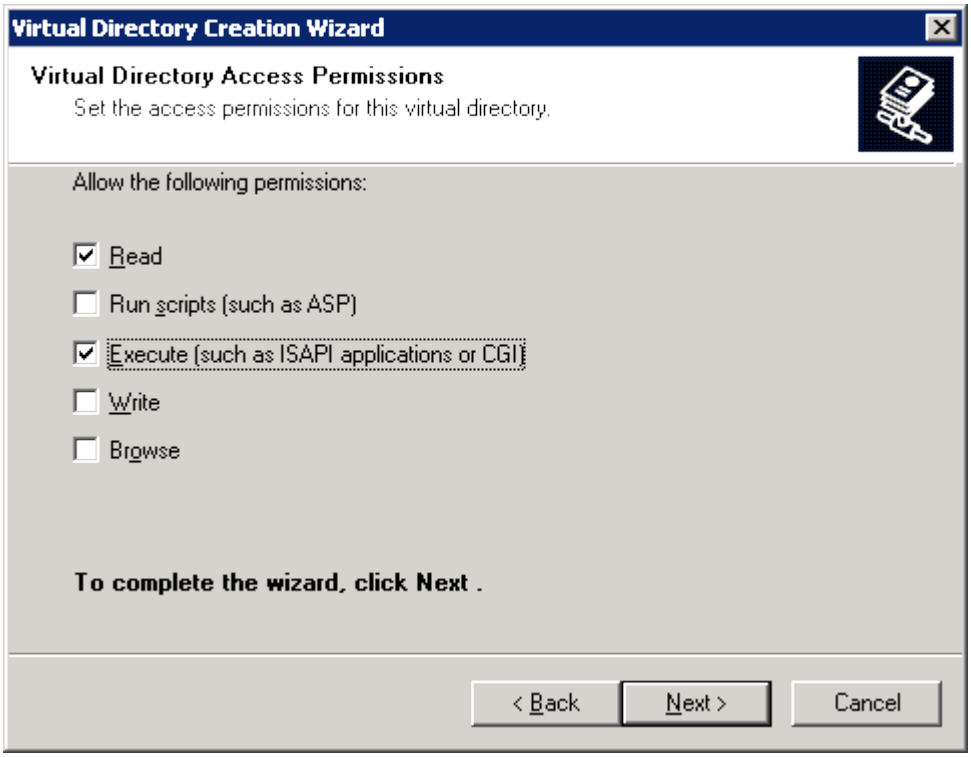
Assign an alias to new virtual directory.



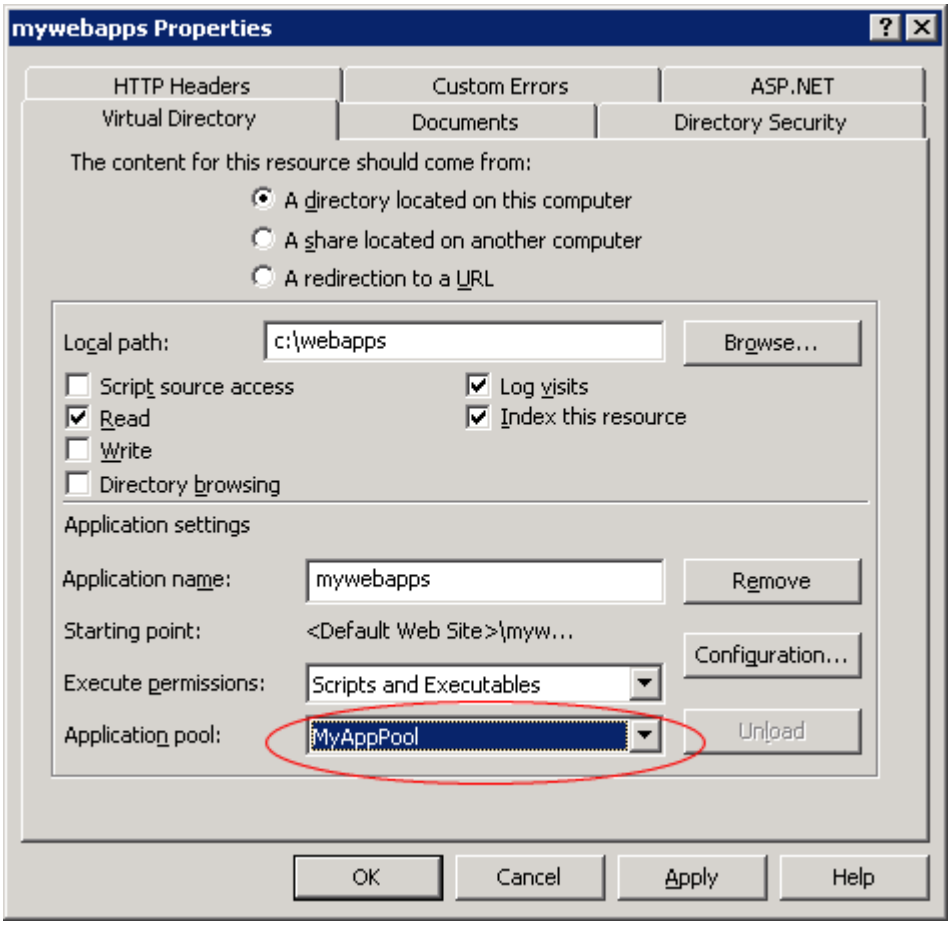
Select the folder where your ISAPI modules are located.



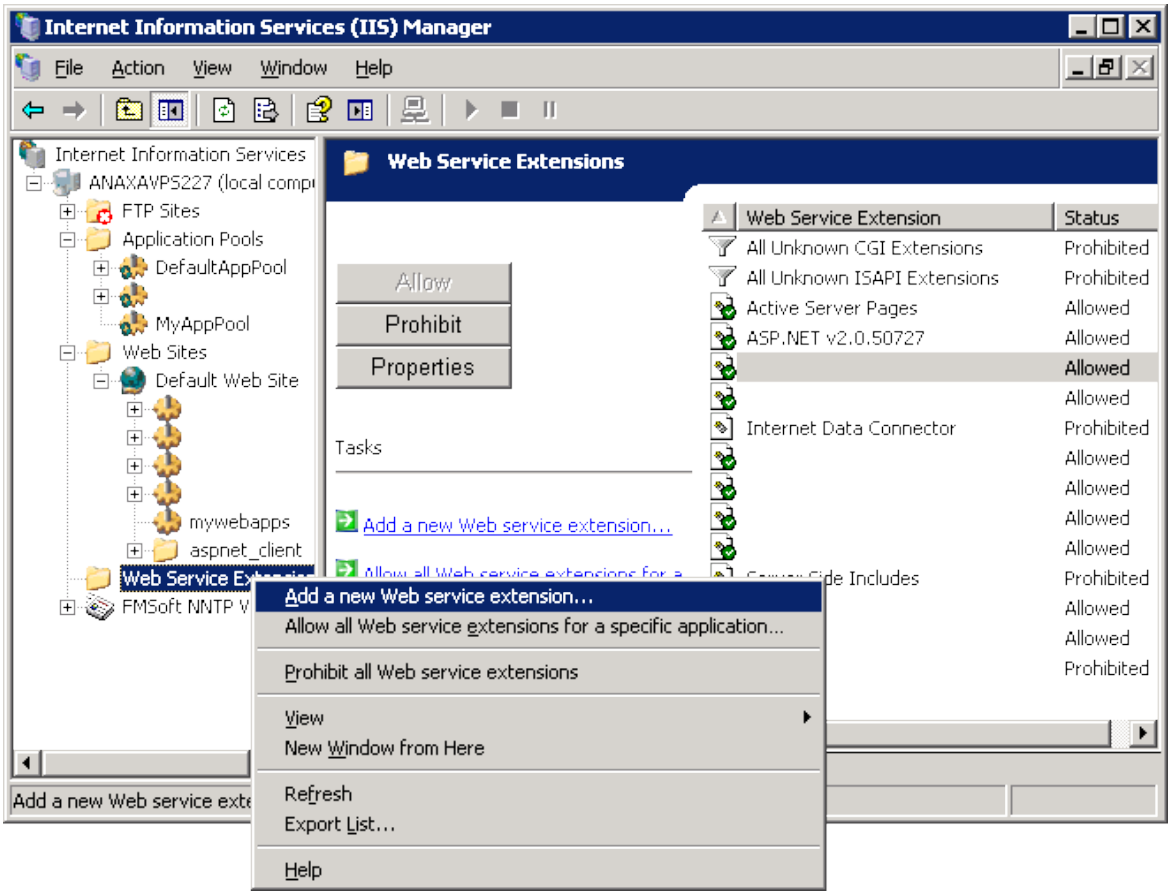
Bu sure to grant **Execute** permission to Virtual Directory.



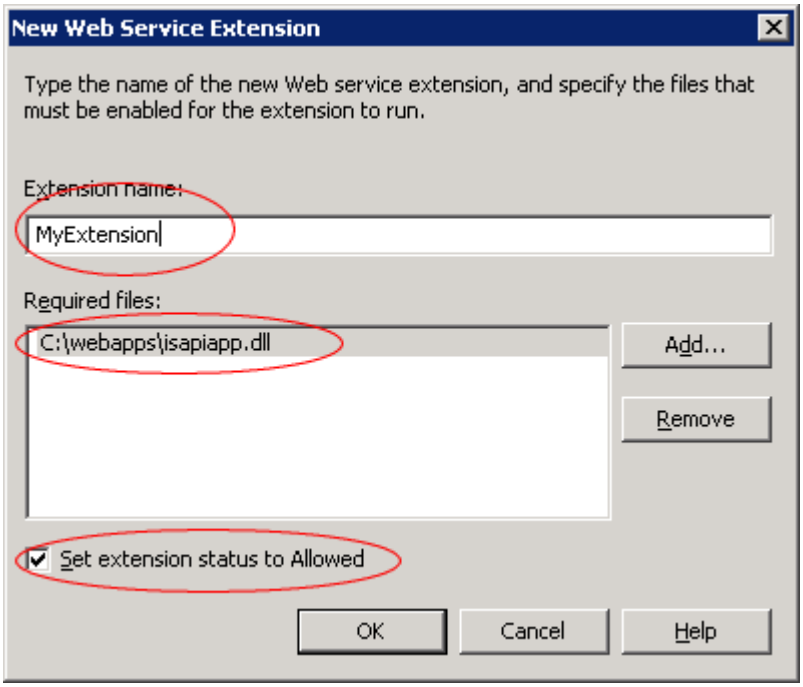
Now open the properties of newly created Virtual Directory and change the default pool to pool you created in first step.



There is one further step in IIS 6. Your ISAPI extenstion must added to list of allowed extensions.

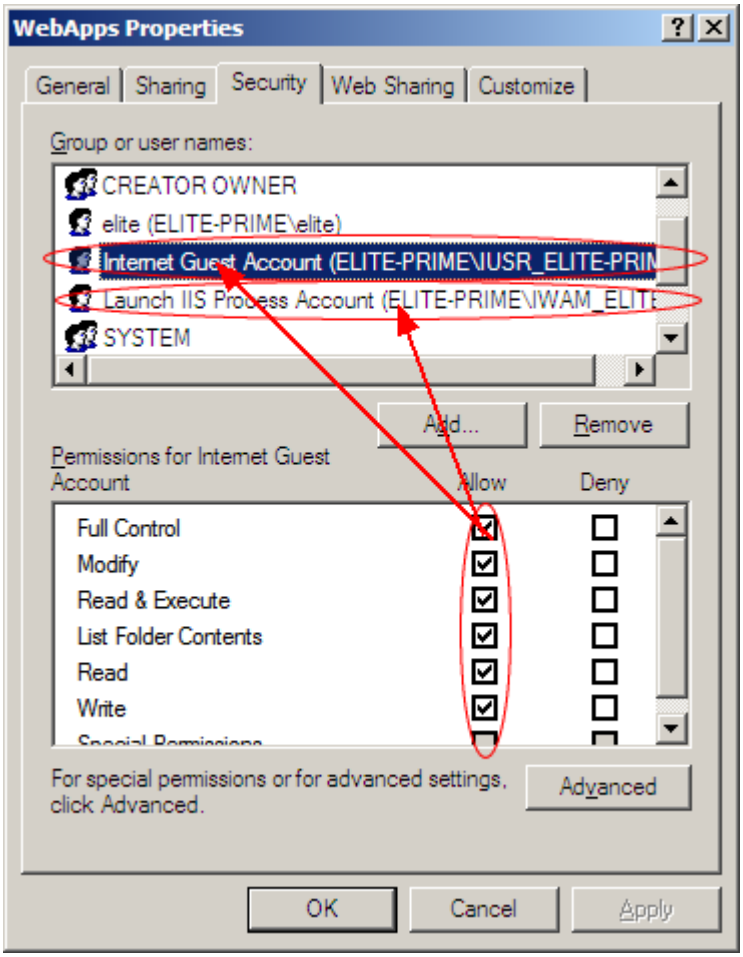


Assign a name to your extension, add the extension's module DLL file and check the **Set extension status to Allowed** option.



After creating a new virtual directory you are ready to deploy your uniGUI server.Copy your ISAPI module and other required files to virtual directory. You must be certain that **ISS** built-in users **IUSR_<ComputerName>** and **IWAM_<ComputerName>** has enough credentials to access your virtual directory and other folders that may be accessed during web application execution.

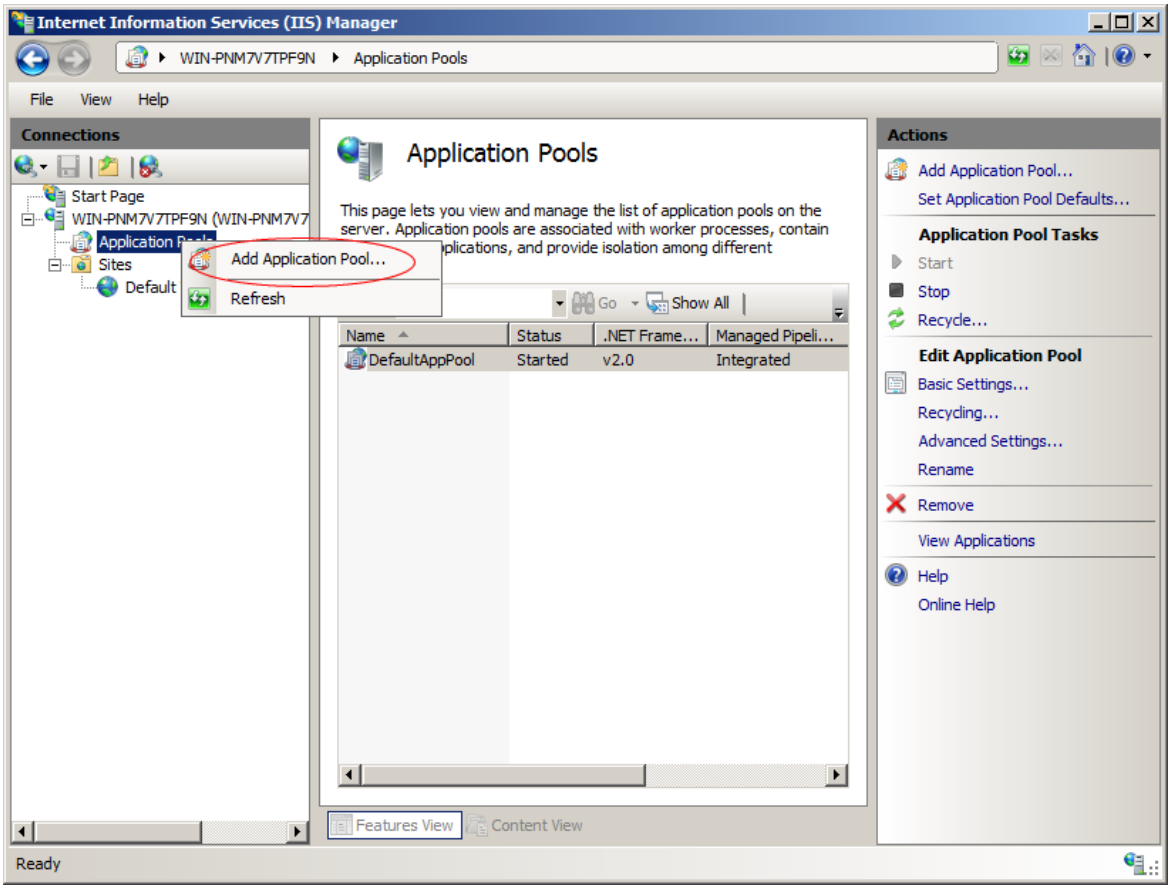
For instance if your ISAPI apps are under folder **C:\WebApps** then you must give full access to



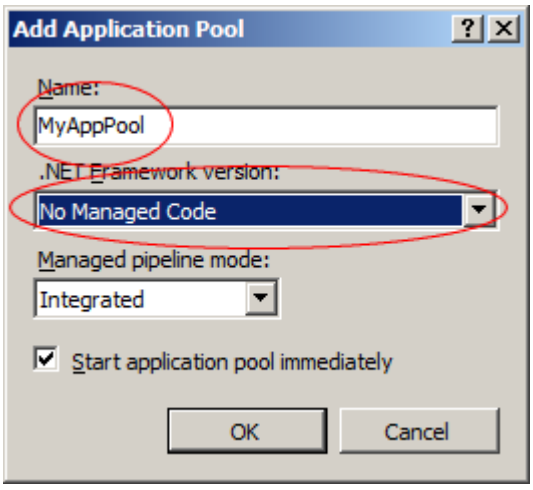
Also refer to Adjusting Paths for more information.

4.3.5.3 IIS 7

Like ISS 6, in IIS 7 first step is to create a new Application Pool.



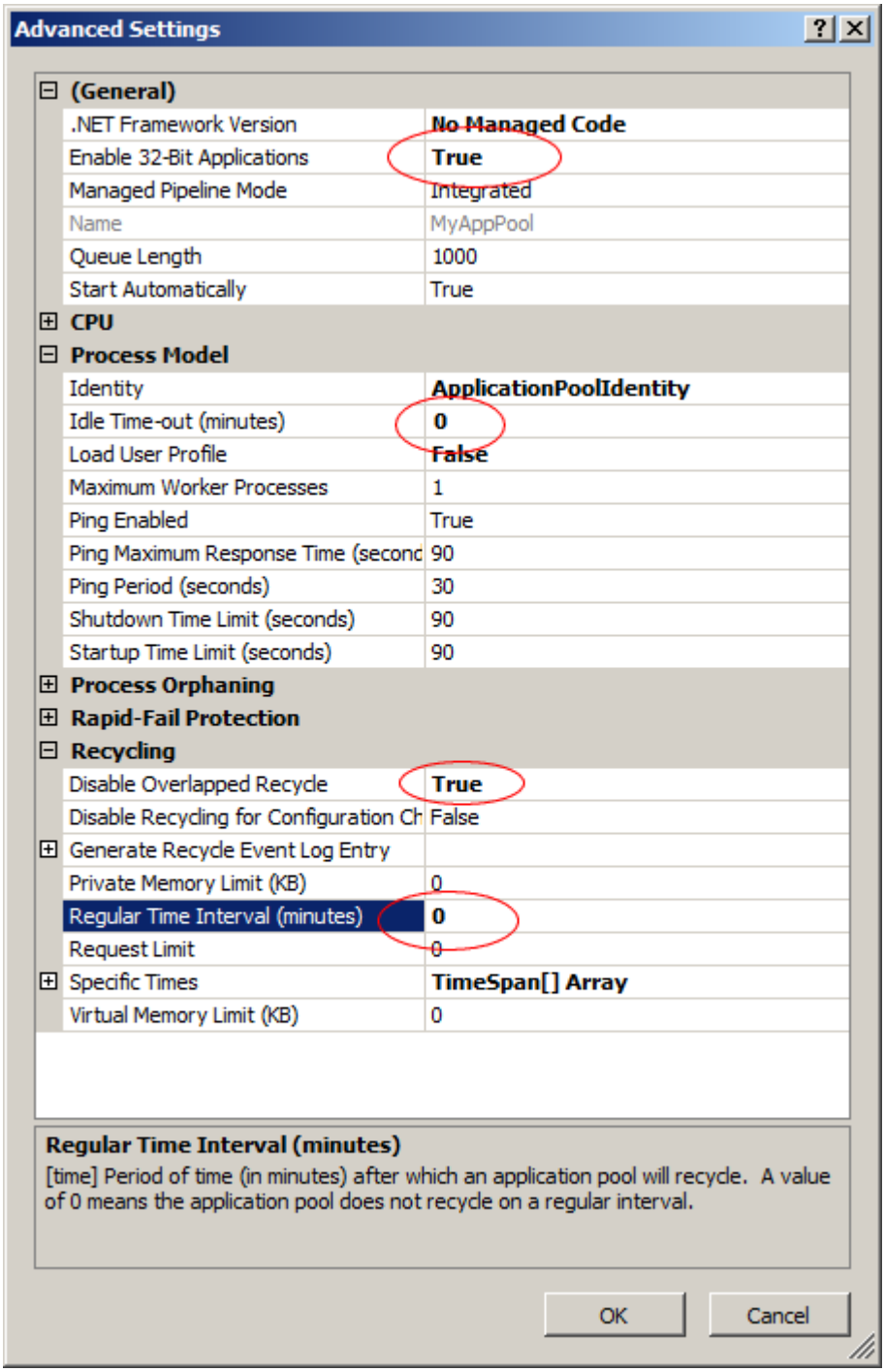
Assigne a name to your pool and select **No Managed Code** option.



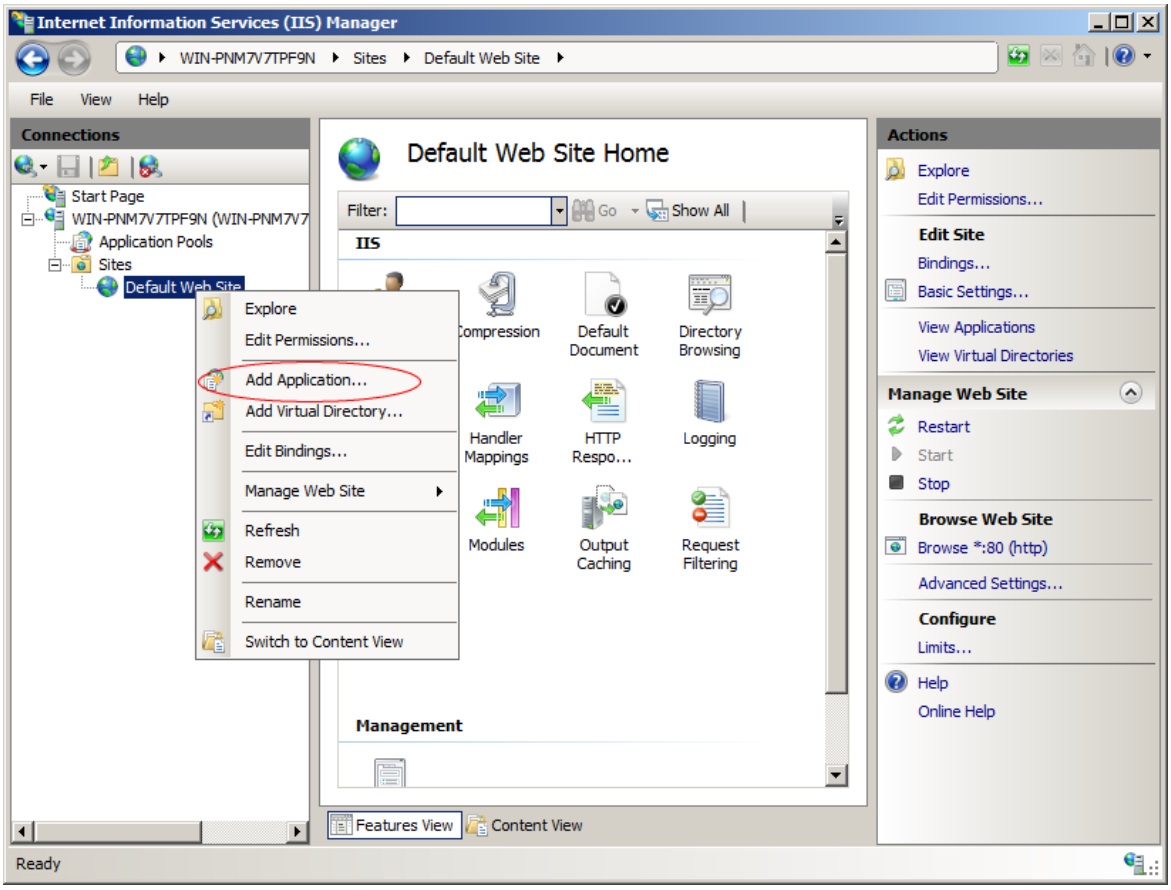
After creating the Pool open Pool's Advanced settings dialog and make the following modifications:

- Set **Enabled 32-Bit Applications** to **True**. (This option is available in 64-bit versions of Windows)

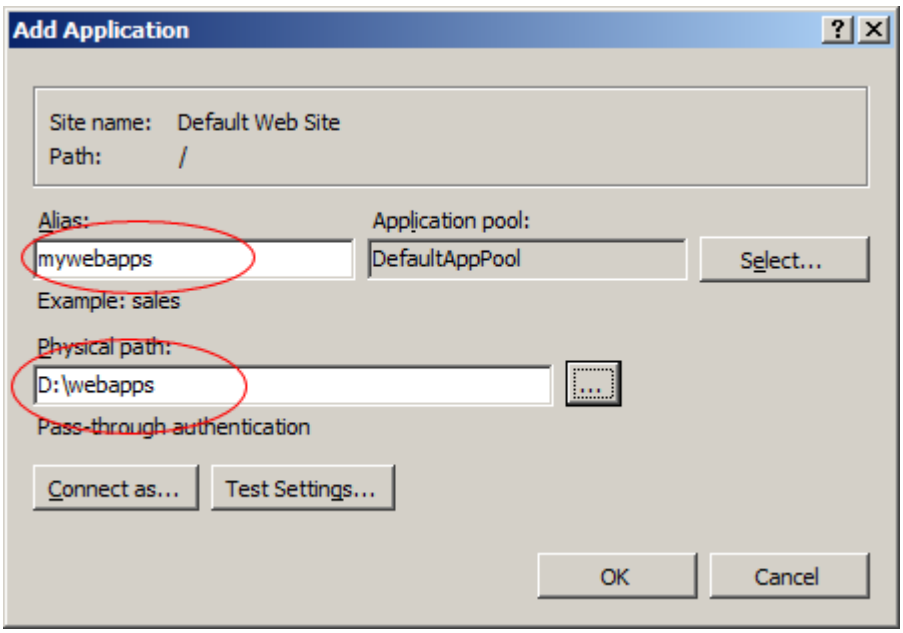
- Set **Disable Overlapped Recycle** to **True**.
- Set **Regular Time Interval** to **0**



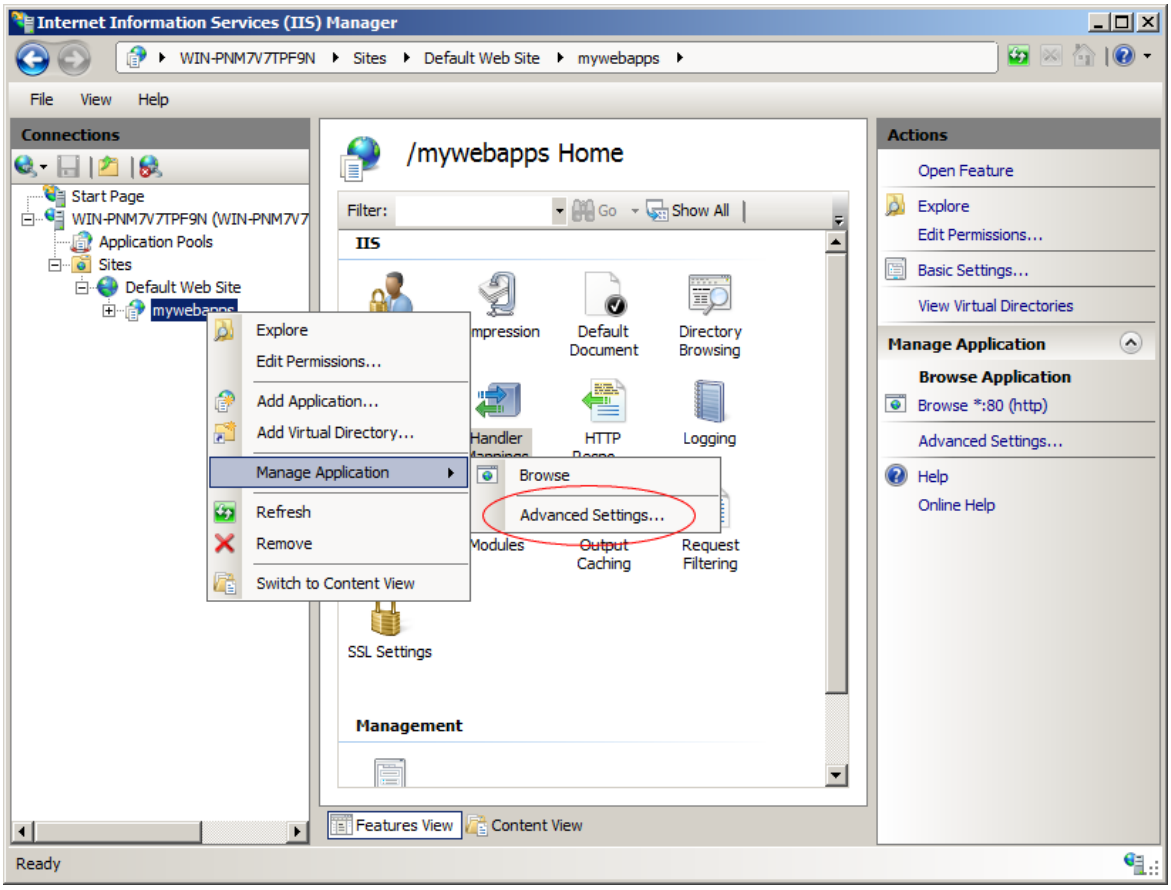
Now add a new Application.



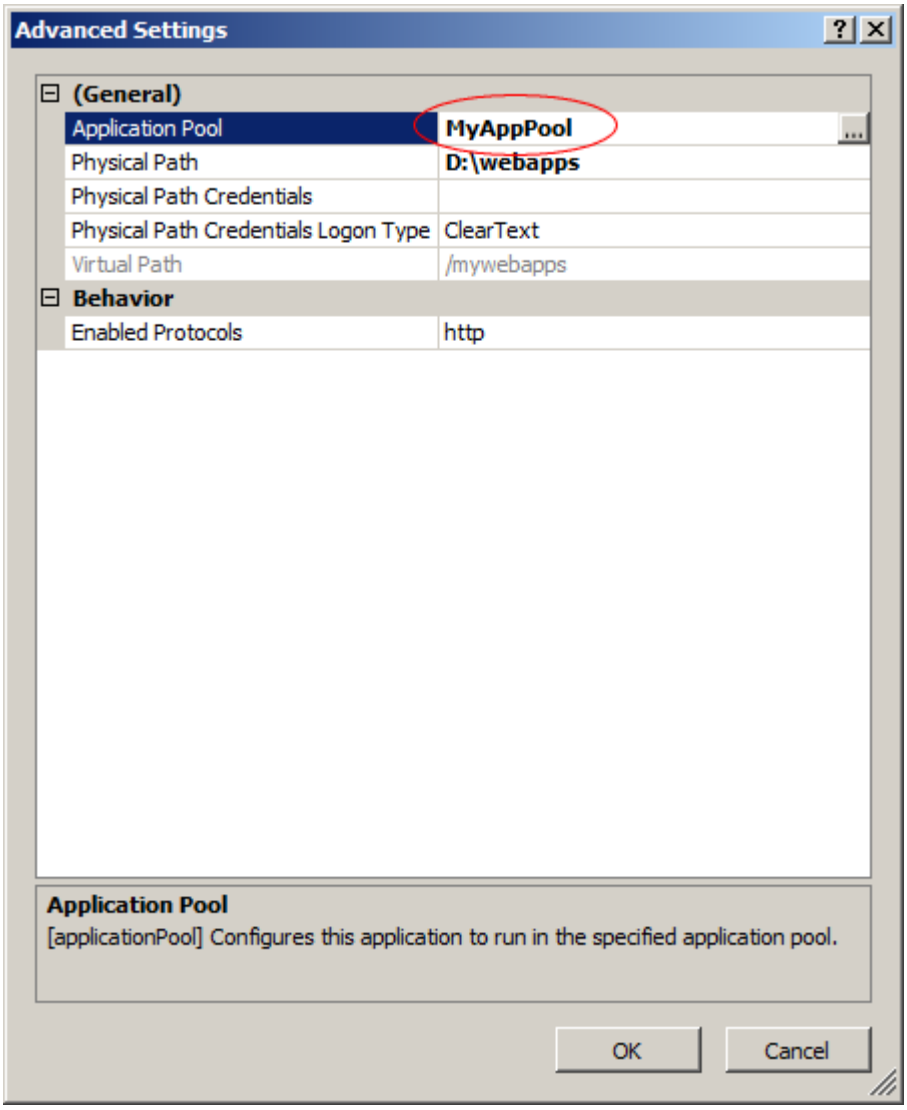
Give it a proper name and adjust the **Physical path**. It is the path where your module DLL files exist.



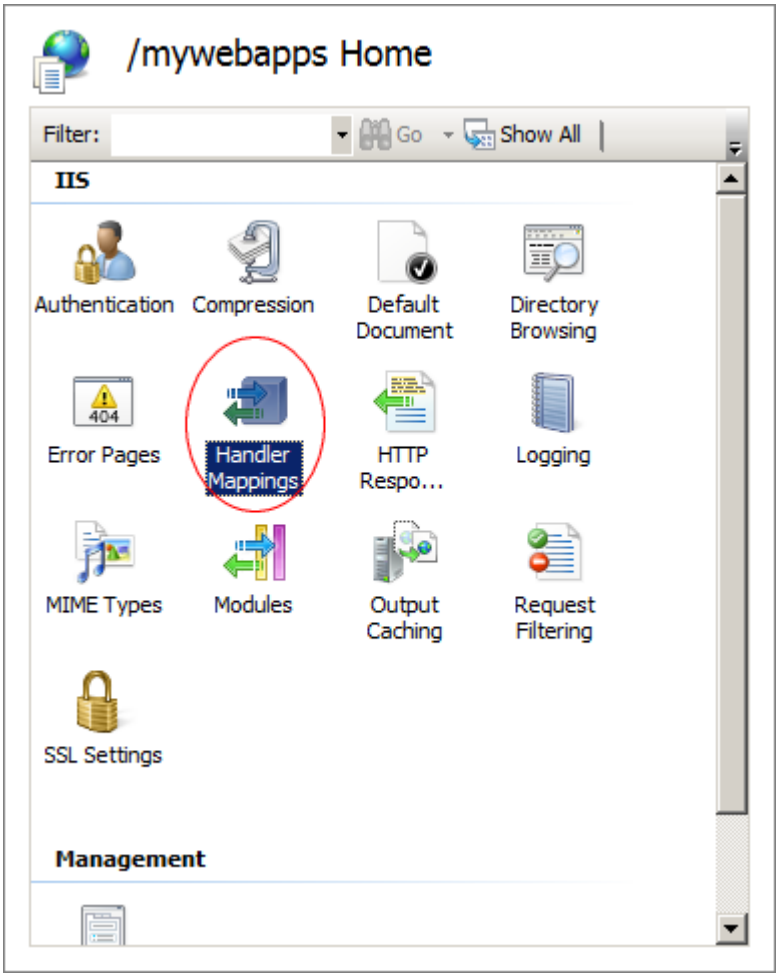
Select Advanced Settings menu and open Advanced Settings dialog.



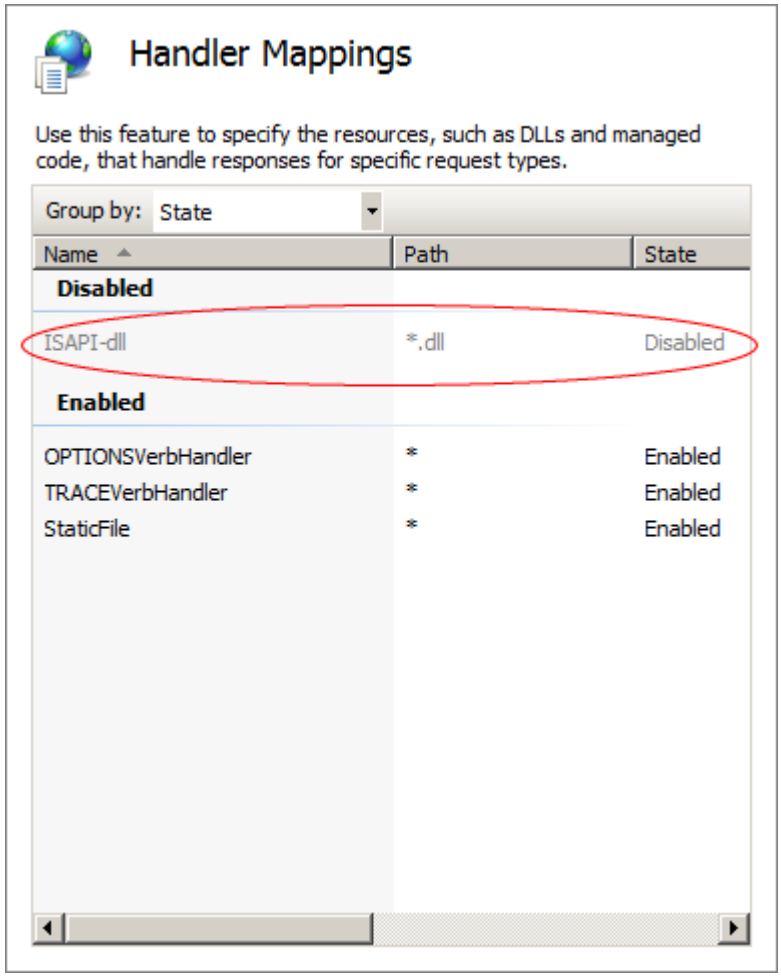
In Advance Settings screen set the **Application Pool** to one you created in first step.



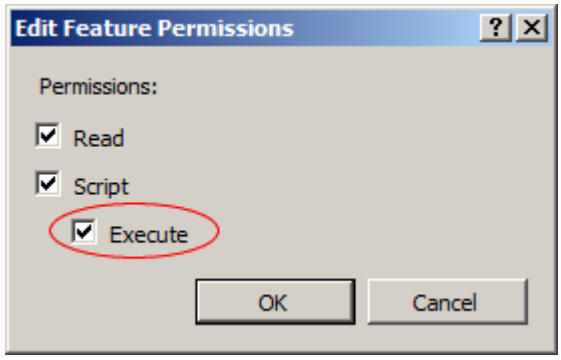
Next step is to adjust the handler mapping for the application you just created.



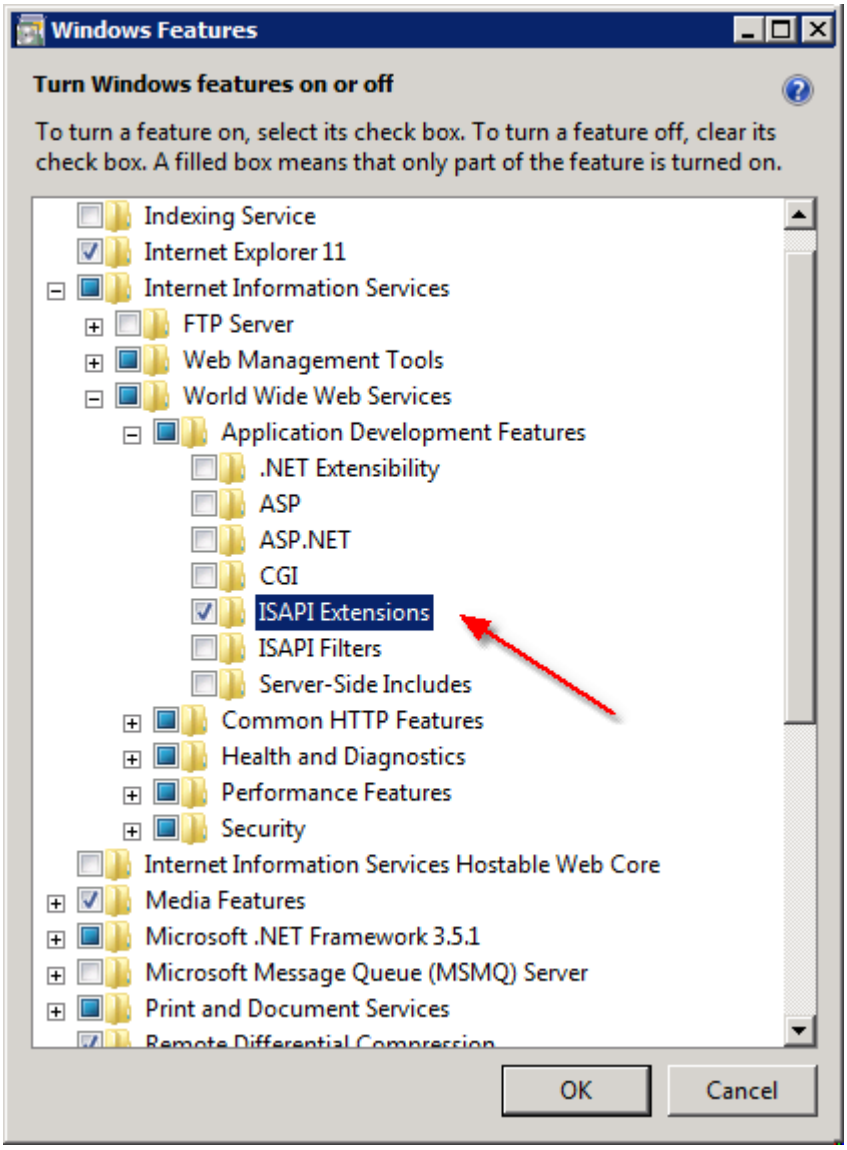
In Handler Mappings screen right-click on ISAPI-dll and select **Edit Feature Permissions**



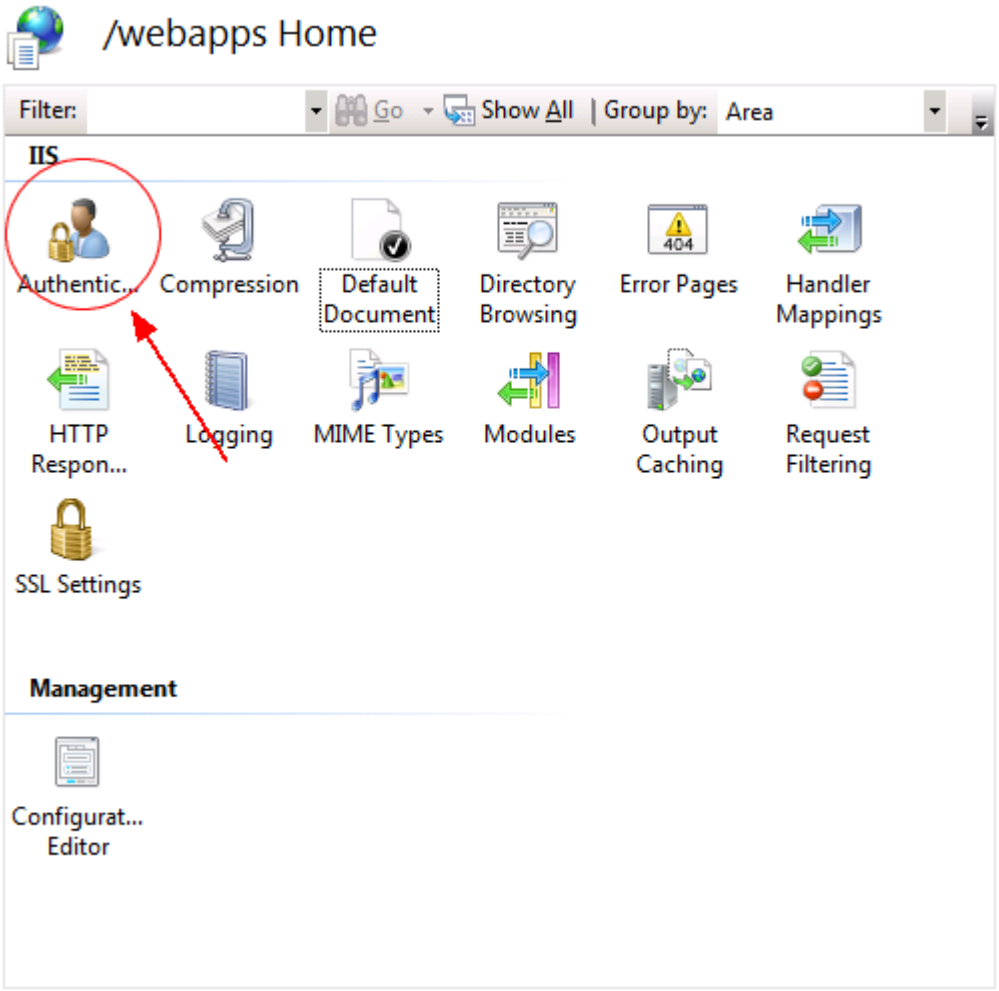
Check the **Execute** option and press OK.



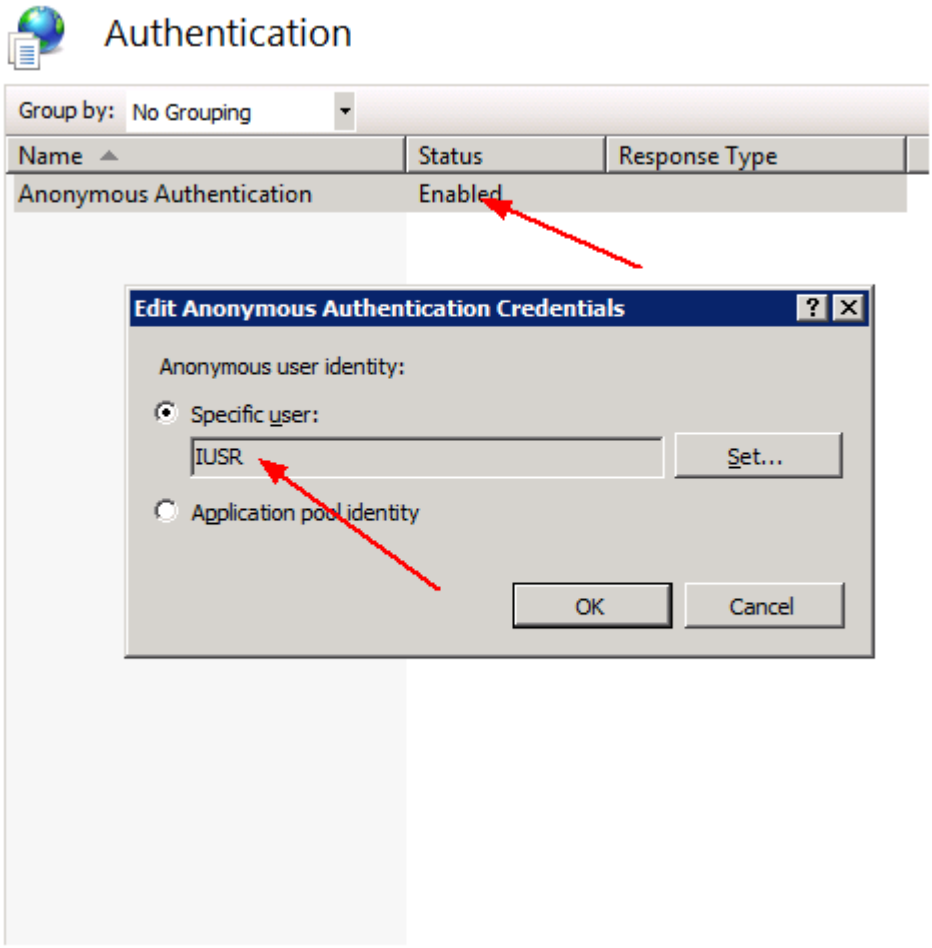
If you don't see ISAPI-dll in handlers list then you must make sure that ISAPI Extensions are installed in Windows Features. Visit Windows Features to check if it is installed.



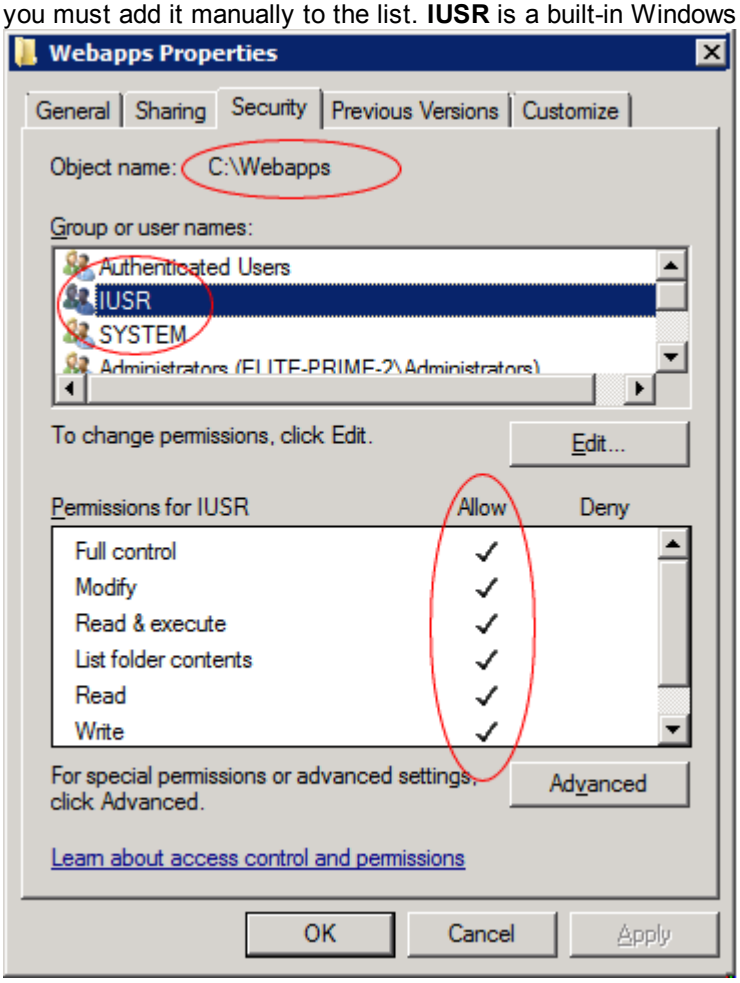
Next step is to check Authentication settings:



Make sure Anonymous Authentication is enabled and the default user **IUSR** is set.



Last step is set ACL bits for web application physical folder. If **IUSR** is not available in the top list you must add it manually to the list. **IUSR** is a built-in Windows user which is created by default.



Also refer to Adjusting Paths for more information.

4.3.5.4 Apache 2.2

Apache 2.2 web server for Windows allows running ISAPI modules. For this, a plugin called **mod_isapi** must be enabled.

Apache doesn't have a visual interface for configuration. You must do some modifications to **httpd.conf** file.

First of all, uncomment the following line:

```
LoadModule isapi_module modules/mod_isapi.so
```

Next you must associate **.dll** files with ISAPI module.

Add the following line

```
AddHandler isapi-handler .dll
```

Next step is add your module directory to Apache directory entries.

```
<Directory "C:/webapps">
    Options Indexes FollowSymLinks ExecCGI
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
```

Finally create a new **Alias** for your directory.

```
Alias /mywebapps "C:/webapps"
```

Be sure to restart your Apache server after making the necessary modifications to **httpd.conf** file.

4.3.6 Windows Service

Windows service is created and deployed like other regular Service applications created using Delphi.

To install service simply type following command at command line*:

```
MyServiceApp -install
```

You can start service from Windows service manager or type the following command:

```
net start ServiceName
```

When you create a new project default value for service name is: **UniServiceModule**
You can change it from **ServiceModule.pas->UniServiceModule->Name**

To uninstall service:

```
MyServiceApp -uninstall
```

**Please note that command line prompt must be started with Administrative rights in order to be able to run above commands.*

4.3.7 SSL Configuration

uniGUI supports SSL protocol for Standalone Server and Windows Service applications. Let's emphasize that built-in SSL support is only for Standalone and Windows Service applications. In ISAPI mode you must configure SSL by configuring your particular ISAPI server; **IIS**, **Apache** and etc.

First step to configure SLL is to obtain the required certificate files. In Standalone and Service modes **uniGUI** uses Indy as the underlying TCP transport layer.

There are three files which you must provide here:

root.pem
cert.pem
key.pem

As you can see files are in pem format. **Pem** files are a human readable **base64** ascii files which can be opened and edited in an editor. A pem file may contain a single certificate or more than one certificate. In order to work with **Indy** each **pem** file must contain only one certificate.

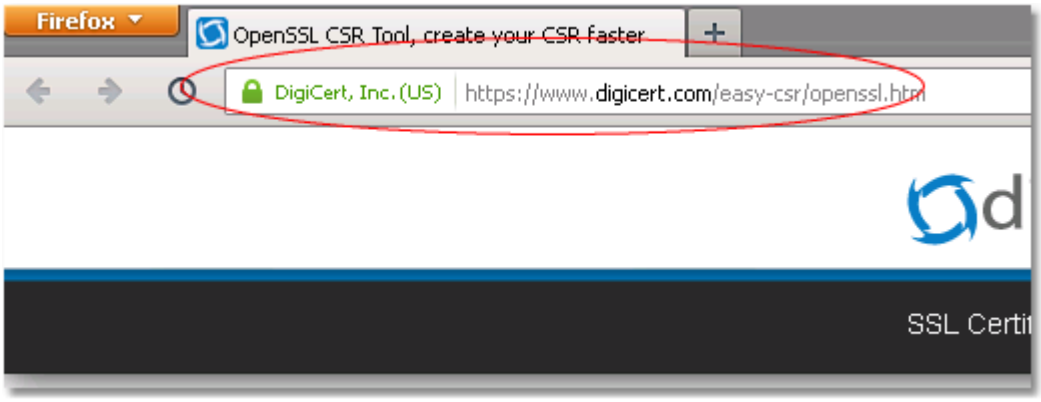
Below sample shows contents of a **pem** file:

```
-----BEGIN CERTIFICATE-----
MIIB8jCCAV+gAwIBAgIQfjGd2Py0qZJGqdkPiRlDdjAJBgUrDgMCHQUAMBAxDjAM
BgNVBAMTBWVsaXRlMCAXDTEzMDYwMjE3NTA0OFoYDzIxMTMwNTA5MTc1MDQ4WjAQ
MQ4wDAYDVQQDEwVlbGl0ZTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAt3pi
pMYHNzUueLZBb1eMrPop6Emta/KLyLaK94vlM1lV/6ITiuFtuSs9gq0s516s2th7
FUkBpgfQvrb+3b9h10WMca8MTbYrLGL+dHqRk4jGt/8GAUeYHkKddk/NeXkZWqaD
3aMdURpTgE2iK/d86C1YdsxqXTxP+Uax/eN4RUECAwEAAaNTMFEwFQYDVR0lBA4w
DAYKKwYBBAGCNwoDBDAtBgNVHREEJjAkoCIGCisGAQQBgjcUAgOgFAwSZWxpdGVA
RUxJVEUtUFJJTUUAMAkGA1UdEwQCMAAwCQYFKw4DAh0FAAOBgQCDSHm54tMh1sPY
aBrpZeZtbt9e1gPZ2B/Gd7U2KGK46yM8OQQ3LlnPaTc96q2ocD9sL3GP1B2itwX/
THOgUX7MpUipfUg6+8te6A7//gjiGyCf/OauJJrHal8p2QPwecGo3YnxUvTCu9gH
+iGE3Yqxv/6YqgDjGnpNdAvvX9gEfQ==
-----END CERTIFICATE-----
```

There are two types of certificate you can use along with your uniGUI server. You can either use a certificate issued by a **certificate authority** or you can use a **self-signed certificate** for development, test and / or private use.

4.3.7.1 Obtain a SSL Certificate from an Authority

SSL certificates are published by companies which are specialized in issuing and hosting SSL certificates. In order to obtain such a certificate you need to make a contract and purchase a certificate dedicated to your company. This certificate will be known to whole web and will contain information regarding your company, the certificate issuer and your web site.



Example for a secured web site

First step is to create a **CSR** (Certificate Signing Request) file. There are several tools to create s CSR file. OpenSSL is best tool which is widely used for such tasks. After you created the CSR file you must send it to your certificate authority and they will verify information you provided in the CSR file. Upon a successful verification they will sign your certificate and issue required certificate files.

Your certificate authority will guide you in process of creating a CSR file and rest of application process.

After a successful application you will receive your certificate files. As described in previous section you need three files to distribute with your server: **root.pem**, **cert.pem** and **key.pem**.

Note: sometimes **cert.pem** and **key.pem** can be issued as a single file. In this case you need to open this file with a text editor and save the individual certificates in separate **pem** files.

4.3.7.2 Generate a Self-Signed Certificate

This kind of certificate is good when you don't need a globally signed certificate issued by a certificate authority such as Verisign. You can use a self-signed certificate for development purpose or for private use in your intranet network or over the internet. You can use OpenSSL to generate related certificate files.

First of all download and install **OpenSSL** Windows binaries from here. After installing open a command prompt and follow below instructions.

We recommend downloading the lite version of the binaries:

Win32 OpenSSL v1.0.1e Light
Win64 OpenSSL v1.0.1e Light

a) Generate a self-signed Root certificate.

If you already have a root certificate installed in Windows you can try exporting it instead of generating a new one. Simply go to **Control Panel** and click the **Internet Options -> Content -> Certificates**. Select the root certificate you want to export. Choose the **base64** format and select folder and file name to export. This will export your root certificate in **.cer** format which can safely rename it to .**pem** and use it in your uniGUI project.

You can also create a root certificate from scratch.

At command prompt issue following command:

```
openssl genrsa -out root.key 1024
```

This will create a new **root.key** file with strength of 1024 bit. Other options are 2048 and 4096. Normally 1024 bit is enough.

If you want to create a root key with a password use below command instead:

```
openssl genrsa -des3 -out root.key 1024
```

Next step is to self-sign the certificate.

```
openssl req -x509 -days 365 -new -nodes -key root.key -out root.pem
```

If your root key is created with a password use below command instead:

```
openssl req -x509 -days 365 -new -key root.key -out root.pem
```

Now you will be prompted to provide several details needed to sign your certificate. You will also be prompted for a password if your root.key file is created with a password in first step.

```
Enter pass phrase for root.key:

You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:TR
State or Province Name (full name) [Some-State]:Ankara
Locality Name (eg, city) []:Cankaya
Organization Name (eg, company) [Internet Widgits Pty Ltd]:FMSoft
Organizational Unit Name (eg, section) []:R&D
Common Name (eg, YOUR name) []:Farshad Mohajeri
Email Address []:info@fmsoft.net
```

Note: **365** is the number days which certificate will remain valid.

This will place a new **root.pem** file in same folder. This file will be used in your uniGUI server.

b) Generate a self-signed key.

Next step is to generate a self-signed key. This step will produce the **key.pem** and **cert.pem** files.

At command prompt issue the following command:

```
openssl req -x509 -days 365 -nodes -newkey rsa:1024 -keyout key.pem -out cert.pem
```

Again, you'll be prompted to answer several questions.

```
Loading 'screen' into random state - done
Generating a 1024 bit RSA private key
........++++++
.....++++++
writing new private key to 'key.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:TR
State or Province Name (full name) [Some-State]:Ankara
Locality Name (eg, city) []:Cankaya
Organization Name (eg, company) [Internet Widgits Pty Ltd]:FMSoft
Organizational Unit Name (eg, section) []:R&D
Common Name (e.g. server FQDN or YOUR name) []:Farshad Mohajeri
Email Address []:info@fmsoft.net
```

To create same key with a password use below command:

```
openssl req -x509 -days 365 -newkey rsa:1024 -keyout key.pem -out cert.pem
```

This time you'll be prompted to enter a password:

```
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

This password will be assigned to SSL->SSLPassword parameter of UniServerModule. (See SSL Configuration)

When all above procedures are completed you will end up with three files named **root.pem**, **key.pem** and **cert.pem** which are required to setup and run your project in **SSL** mode.

4.3.7.3 Configure SSL Parameters

Copy all three **pem** files to same folder your **uniGUI** server executable resides.

You also need **OpenSSL** standard **DLL** library files.

```
libeay32.dll
ssleay32.dll
```

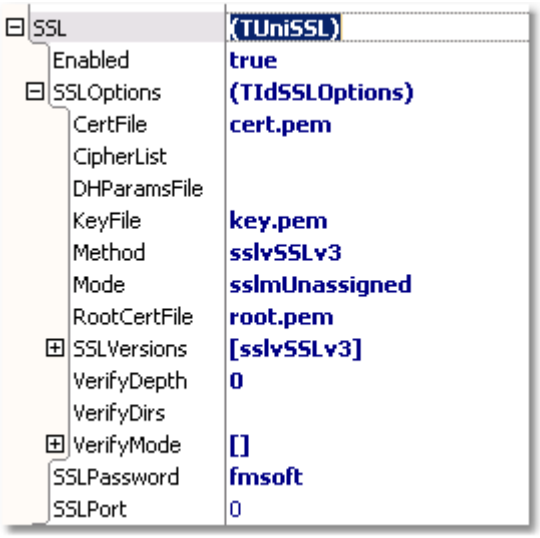
These **DLL** files are distributed as a part of **OpenSSL** installation. You can also find the most recent version of **OpenSSL DLL** files under below folders:

```
[UniGUI Installation Folder]\..\Framework\uniGUI\SSL\dll\x86
[UniGUI Installation Folder]\..\Framework\uniGUI\SSL\dll\x64
```

Depending on how you deploy your app you can copy them under Windows system folder (such as `C:\Windows\System32`) or put them in same folder as your **uniGUI** server.

Note: There is a chance that above dll files are already installed by other programs and already present in your system path. In this case you must make sure that most recent versions of dlls are installed.

Next step is to open your project's **ServerModule** and make the following changes:




```
SSL->Enabled = True
SSL->SSLOptions->CertFile = cert.pem
SSL->SSLOptions->KeyFile = key.pem
SSL->SSLOptions->RootCertFile = root.pem
```

If you have protected your key files with a password then you need to assign it here:

```
SSL->SSLPassword = [Password]
```

Next parameter to confgure is the SSL port. You can leave it to default value which is **0.** In this case you can access your server through port which is configured in **ServerModule -> Port** which default value is 8077.

```
https://localhost:8077
```

Or you can use the default port for SSL whch is **443**. Since **443** is the default value you can omit the port number in address:

```
https://localhost
```

Normally you only want to use **https** protocol for a secured web application, but if for any reason you need both **http** and **https** protocols for same site you can enable them by assigning different ports to each protocol.

Consider the following configuration:

```
ServerModule->SSL->SSLPort = 443
ServerModule->Port = 8077
```

In above configuration you are able to access standard **http** protocol from port **8077** and access **https** protocol from port **443**.

If you choose the default port of **443** for SSL make sure neither **IIS** nor any other web server software is not listening on this port.

4.4 Stress Test Tool

4.4.1 Introduction

Scalability is one of the major concerns when developing web applications. Each web application must be able to serve multiple users. Number of users can vary depending on type of application. An application designed for a small intranet may be targeted by 10-50 concurrent users at most while an internet web application may be used by more than 500-1000 simultaneous users.

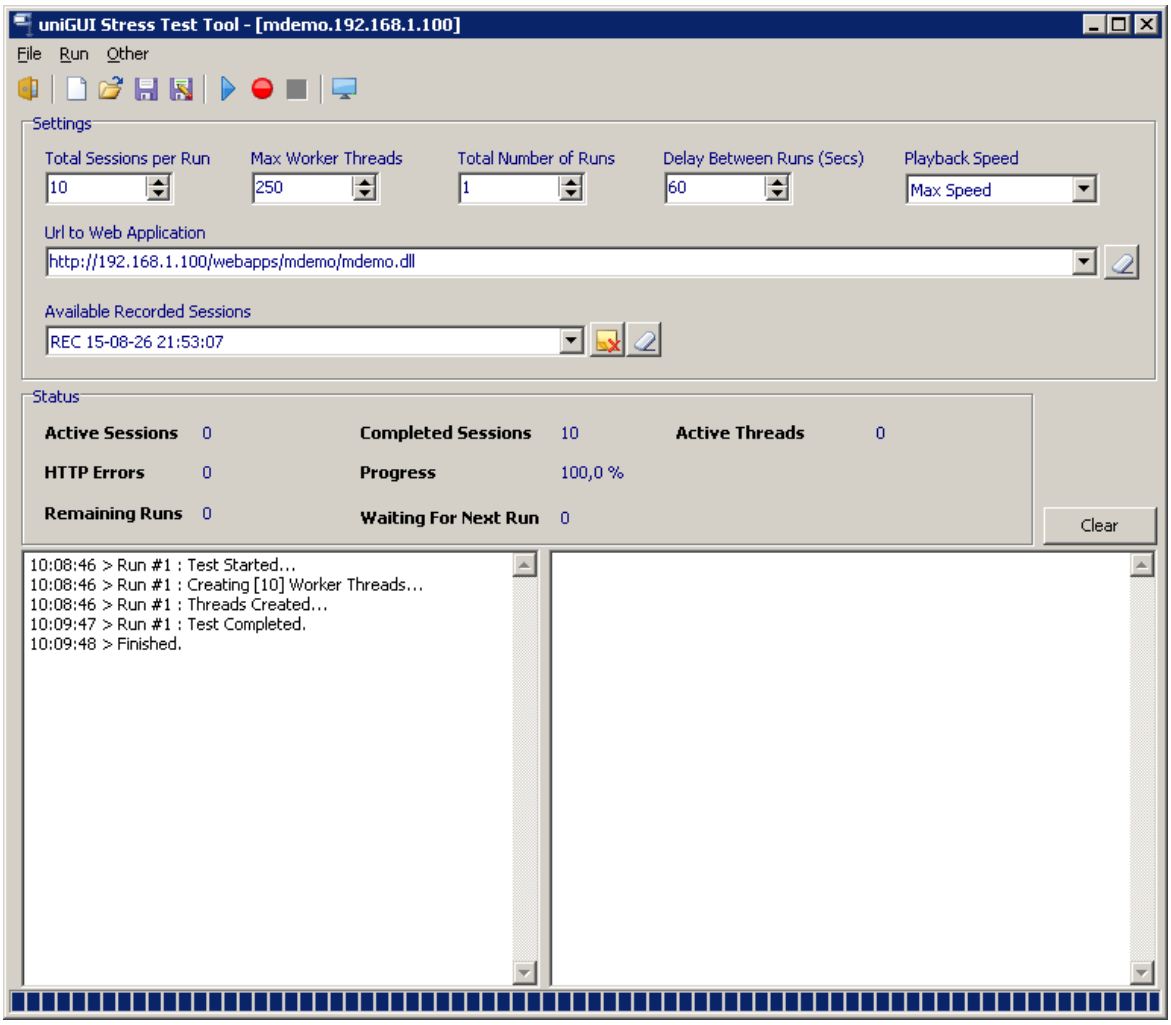
In order to make sure your web application is ready for deployment in multi-user production environment several tests must be performed. Firstly, application must be analyzed for its resource usage. For a desktop app where only a single instance of your app is running resource usage will not be your main concern because your application can use all of resources available in system. On the contrary, a web application server is designed to serve multiple users and each user can be dedicated only a fraction of available resources.

Scalability simply means ability of your project to scale up when number of sessions increases in a real world system. A scalable web app should use several techniques to manage and use system resources in most efficient way possible. For example, if each of your sessions consumes ~1000 KB of memory (~1MB) in this case a web app running 400 sessions will require at least 400 MB of physical/virtual memory to run properly. However, if each session consumes more than 10MB then it is easy for your app to run out of memory when there are more than 100 active sessions. Other system resources such as disk space, database connections and etc. should also be taken into account.

For developers it may not be an easy task to simulate production environment on his/her desktop. Your app maybe running smoothly when there are only a few sessions, but when number sessions start to pass above a certain threshold several resource related issues and scalability problems may arise. To deal with such scalability issues **uniGUI** is equipped with a very useful named **Stress Test Tool** which aims to simulate a multi-user environment right on your desktop. **Stress Test Tool** is located under **..\uniGui\Utils\StressTestTool** folder and it is deployed with full source code so it can be customized by developers if needed. This tool introduces many advanced features which enables developers to simulate a production environment under heavy load.

4.4.2 Usage

Here is the main form of stress test tool:



Using this tool you can record a web session and then playback to simulate multi-user behavior in production environment. Stress test tool an stores each test environment in a project file which can be used later to repeat the test. A project saves all of your recorded sessions and test parameters. It is good to create a separate Stress Test project for each uniGUI project. uniGUI Stress projects are stored under **Projects** folder with a extension of (*.uprj)

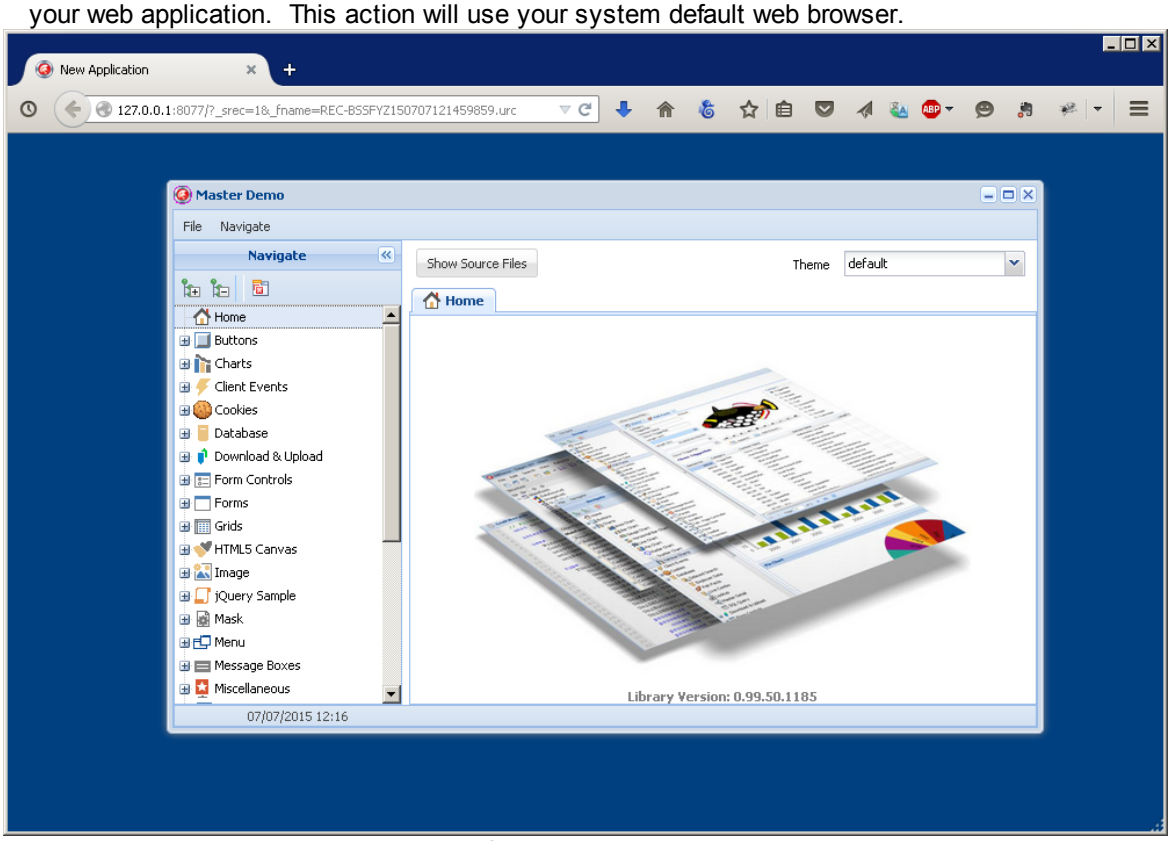
4.4.2.1 Recording a Session

Recording a session is an easy task.

- First of all, you must compile your server with **ServerModule->Options->soAllowSessionRecording = True**. This parameter must be set to False once your app is ready for production.
- Build and run your uniGUI server.
- In **Stress Test Tool** set **Url to Web Application** to Url address of your server. If your server is

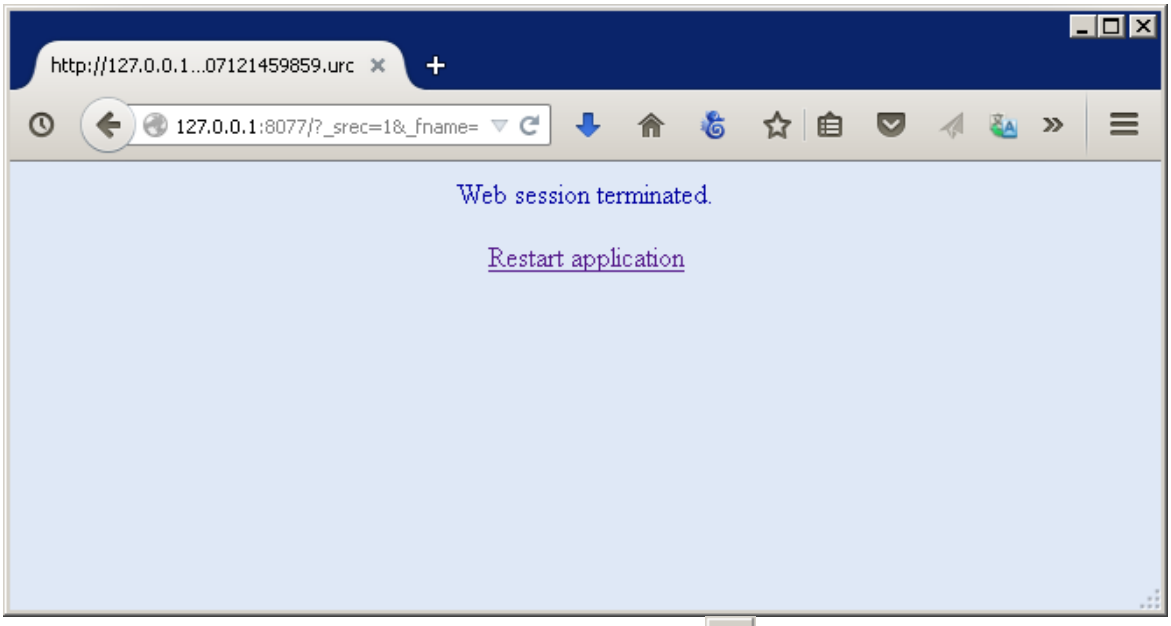
running in same machine a typical address would be http://127.0.0.1:8077

- Press the Record  button. This will launch a new browser window which will show main form of your web application. This action will use your system default web browser.



Session Recoding

- At this stage you can start using web application by navigating in menus, forms, grids and etc. All of your actions will be recorded.
- You can continue recording this session until you think all required menus, forms, grids and functions of your app are fully visited.
- When you are done you can terminate your web application.



- Now switch to Stress Test Tool and press the Stop button  .
- This finish your record session and adds it to list of available records for this project.
- You can save your project using Save button  .

Important: A session must be recorded in a way that session output is always predictable i.e. it will always produce same HTML content. For example, pressing **Button1** should always open **Form1** and selecting menu item **User1** should always show a frame named **TUserFrame**. Normally, it is the case for 99% of web applications where each user action results same output. There are certain cases where user action may not produce a predictable result. Inserting rows in a DBGrid is an example for this case. When user inserts a new row it creates a new grid row and also a new grid page after certain number of rows are inserted. This means that each inserts produces a different JavaScript content which is not predictable and it totally depends on current number of rows in the attached database table. For this, it is recommended to avoid inserting data into a DBGrid while recording a session.Likewise deleting or editing a row may not run properly during session playback. Consider that your table contains 100 rows and you delete a single row while recording. When you playback 100 instances of same session will try to delete same row while only one session can actually delete it.

4.4.2.2 Running the Stress Test

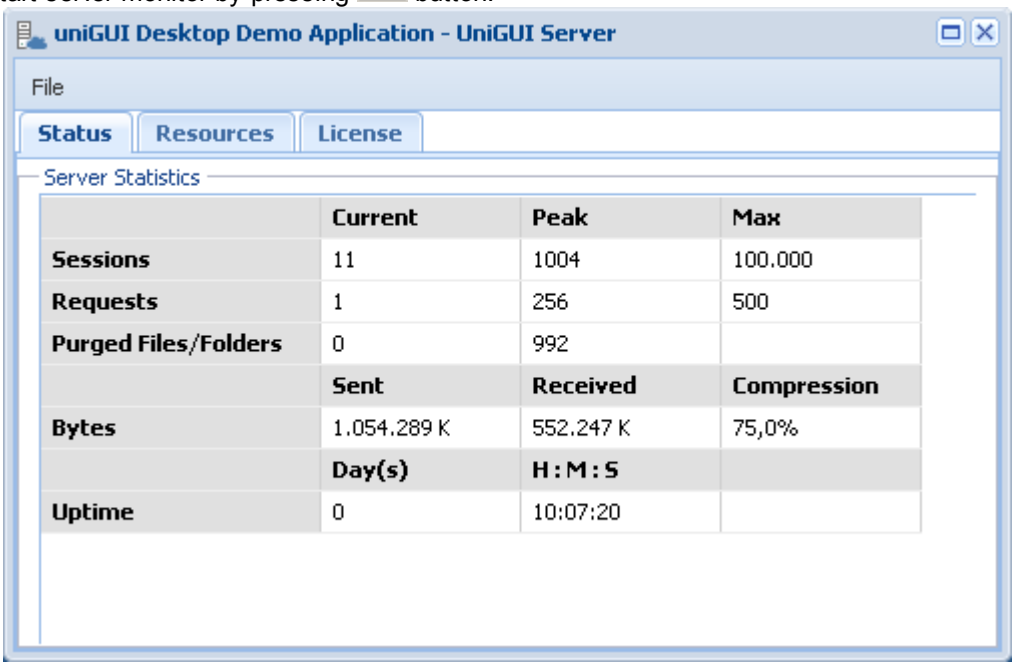
Running a stress test by playing a previously recorded session is an advanced feature which will help you to simulate a real time multi-user environment. After you have one or more recorded sessions you can proceed with actual stress test procedure.

Stress test tool will play back a session by sending sequences of recorded Ajax requests to server. This will simulate an environment where many users are using the web application. It is important to know that play back process does not run your web application in a real browser window. The simulation is performed by sending previously recorded Ajax packets to server in the exact order generated while recording a session.

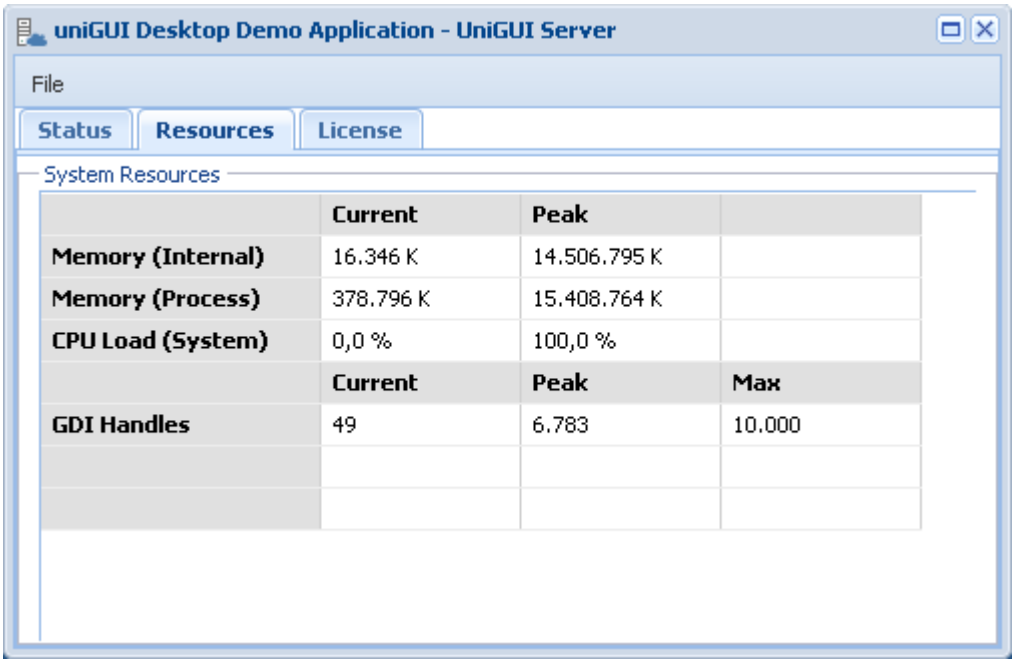
To run a stress test you must follow below steps:

- **Make sure that you don't run the Stress Test Tool in IDE and in debug mode. Running in debug mode will highly reduce speed and performance of the Stress Test Tool. It is better to compile it and run the executable directly.**
- **To record and playback a mobile session you must explicitly add '/m' to end of the application Url.**

- Load a saved project or record a new a session if there are no recorded session available. (see previous section to see how sessions can be recorded).
- Adjust various test parameters:
  - **Total sessions per run:** This is the number of sessions which will be created in each run. It is proper to start with a low value and increase it gradually. This value should be a value lower or equal to server's *ServerModule->ServerLimits->MaxSessions* property.
  - **Max Worker Threads:** Number of worker threads used to communicate with the server. This should be a value lower or equal to server's *ServerModule->ServerLimits->MaxRequests* property.
  - **Total number of runs:** Total number of times that recorded session will be played.
  - **Delay between runs:** Number of seconds to wait before a new run starts
  - **Playback Speed:** Using this parameter you can adjust the speed at which session is simulated. If you choose Max Speed no delay will be inserted between Ajax calls so your session will be played at max speed. You can also choose Real Time which will try to simulate playback close to the original timing of the recorded session.
- Start server monitor by pressing  button.



- Now you can press Start button  which to start the actual stress test.
- Start monitoring your server by analyzing various real time information displayed in server monitor:



- **Sessions:** It shows number of sessions that are running in your uniGUI server. Peak value shows maximum number of sessions that where running during server lifetime.
- **Requests:** Number of pending Ajax requests which are currently being processed by the server.
- **Purged Files/Folders:** These are files or folders which no longer belong to a session, but not erased yet. Framework will gradually delete purged files and folders in a background thread. Under normal conditions this value should eventually become zero.
- **Bytes:** Number of bytes Sent/Received during server lifetime.
- **Uptime:** Time passed since server has been started.
- **Memory:** It is one of the vital information which you need to constantly keep an eye on while performing the stress test. It will show total amount of memory your server application is consuming. It is important especially when your server is a 32-bit app. While 32-bit application can consume up to 2 GB of RAM in theory, in practice any memory usage value above 1.0 - 1.1 GB means that your application is in danger zone. That happens because memory used by uniGUI app will be fragmented over time. For this and for other reasons Delphi memory manager can not use a total of 2.0 GB of RAM for a 32-bit uniGUI server. When your app runs out of memory it will start logging "Out of Memory" messages and user will also see this warning when a new session is started.

This is a serious situation and may produce unpredictable results which will eventually lead to a server crash. If you are concerned with memory usage and your app will use more than 1GB of RAM it is better to target your server for 64-bit platform. In a 64-bit server it is very unlikely to get an "Out of Memory" error because even when physical RAM is consumed Windows will switch to virtual memory which is limited by swap space available on your hard drive. So in a 64-bit server you can consume memory much larger than the available physical memory. Of course, it must be noted that virtual memory is much slower than physical memory, so in a 64-bit uniGUI server when your system runs out of physical RAM you will notice a considerable drop in web application's performance and responsiveness.

  - **Memory (Internal/Process):** Internal memory is amount of memory internally used by the uniGUI application. Process memory is the total memory consumed by Windows process which is the uniGUI application. For example, if your app is an ISAPI dll main process which is an ISAPI worker process will use more memory than internal memory used by the app itself. If your app relies on external dll libraries such as Midas, again memory used by process will be higher than memory used internally.

- **CPU Load:** This will show amount of CPU power currently being used. If your CPU has multiple cores (which is the case for all modern CPUs) this value will show total CPU usage for all CPU cores. An idle uniGUI server will use very low CPU ticks so when both OS and uniGUI app are idle CPU usage should not be more than 1%.
- **GDI Handles:** GDI handle is a limited system resource which is limited by OS. uniGUI is optimized in a way to keep GDI handle usage at a very low level. If your app uses components which rely on GDI handles you must check this value to see if it remain within a safe range. A typical uniGUI app will not consume more than 100 GDI handles when it is in idle state (there are no pending requests) even when there are hundreds of active sessions. uniGUI is designed in a way so a GDI handle is returned to OS as soon as it is not needed. If you see a constant increase in GDI handle then you must look for components which may leak GDI handles or overuse them. When your app runs out of GDI Handles it will start logging **EOutOfResources** exceptions in your log file. This is a serious problem which will eventually lead to server crash.

- Another important step is to monitor your web application log files. Stress test runs in background without any GUI output, so log file is the only way to see if your app runs properly. It is also the only way to see any error logs generated while running the stress test. Log files are placed under **.\logs** folder where your application's binaries are located. For each calendar day a separate log file is created. You must open related log file in a text viewer/editor and see if there are any unexpected logs such as Access Violations or other important Exceptions. By carefully analyzing the log file you can have a better idea about your application stability and scalability. If you need a more detailed log you can use a more advanced debugging tool such as **EurekaLog.**
- **Important:** You must take this into consideration that each stress test runs same recorded session in simultaneous threads for multiple times from same URL, so it is important to set the **ServerModule->SessionRestrict** parameter to **srNone** while you are testing your server.

4.4.2.3 Server Flood Protection Considerations

Apparently, **Stress Test Tool** will send lots of requests in a very short periof of time. All these requests are sent from same IP to your uniGUI application which is being tested. Your server OS may interpret these high volume of requests as a flood DOS (Denial Of Service) attack and start blocking such requests. It may happen especially if you run **Stress Test Tool** from a remote PC with **PlayBack** speed set at **Max Speed** for a long period of time. Since flood detection algorithms varies from OS to OS and highly depends on particular configuration and used server software it is hard to predict when such blockages may occur. When server starts blocking requests you will notice lots of HTTP error messages in right side memo in Stress Test Tool. This shows that requests are being blocked and are not reaching their target.

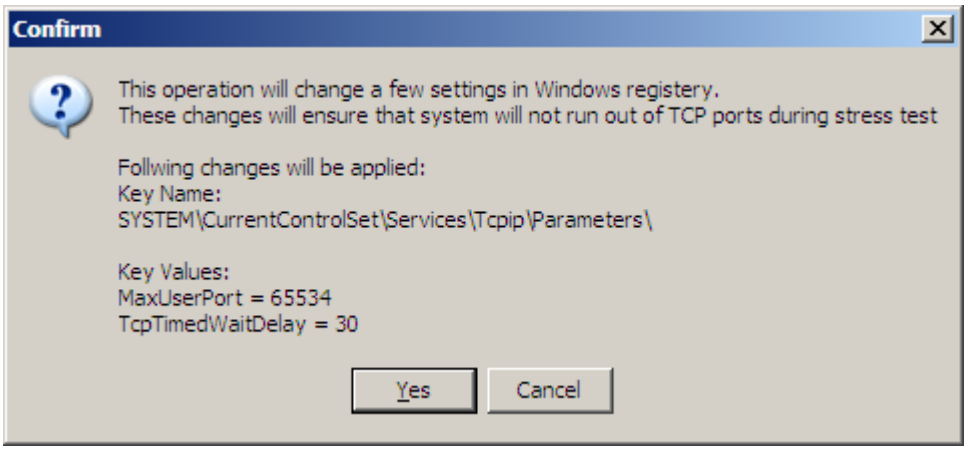
For example, behavior will be different among running uniGUI app as ISAPI module in IIS, standalone EXE and Windows Service. While running the **Stress Test Tool** from a remote PC may trigger flood protection, running both uniGUI app and **Stress Test Tool** on same desktop should work without triggering any flood protection mechanisms.

Additional to server OS some third party protection software such as **Kaspersky**, **Symantec** and etc. may add additional layers of protection to your server IP traffic which may disallow fast and continuous requests coming from Stress Test Tool. In such cases you need to slow down playback speed by setting it to **Real Time** and/or putting a longer delay between **Runs** to allow the requests pass through the anti-flood detection layers.

4.4.2.4 Additional Settings

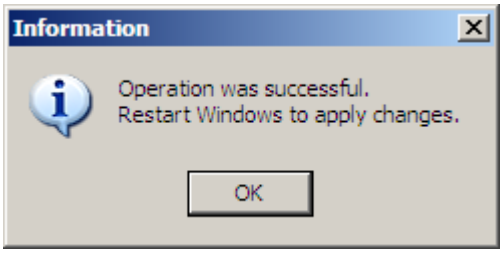
By nature **Stress Test Tool** creates hundreds of HTTP requests per second to perform a heavy stress on subject server. This may go beyond the limits of OS which is running the **Stress Test Tool**. One of these limitations is TCP port limit which is limited to a certain value by OS by default. In order to make a smooth run of **Stress Test Tool** we need to adjust the TCP port limit and increase it to a higher value. This requires a few registry setting in Windows Registry.

Good news this that **Stress Test Tool** can automatically make this change in registry for you. All you need to do is selecting **Other->Adjust Registry Settings** in **Stress Test Tool**.



Above dialog will be opened and it will give a brief information about the settings and will ask you to confirm the operation. These changes are totally harmless and will increase TCP port creation limit on your OS.You must make sure that you have started **Stress Test Tool** as **administrator** to be able to make this change.

A successful operation will show below message and ask you to restart your PC to finalize the operation.



Index

- D -

DataModule 21

- E -

ExtRoot 45

- L -

LoginForm 21

- M -

MainForm 21
MainModule 21, 27
ModalResult 25
mrCancel 25
mrOK 25

- R -

RIA 4

- S -

ServerModule 21, 27

- U -

uniGUI 4

Endnotes 2... (after index)