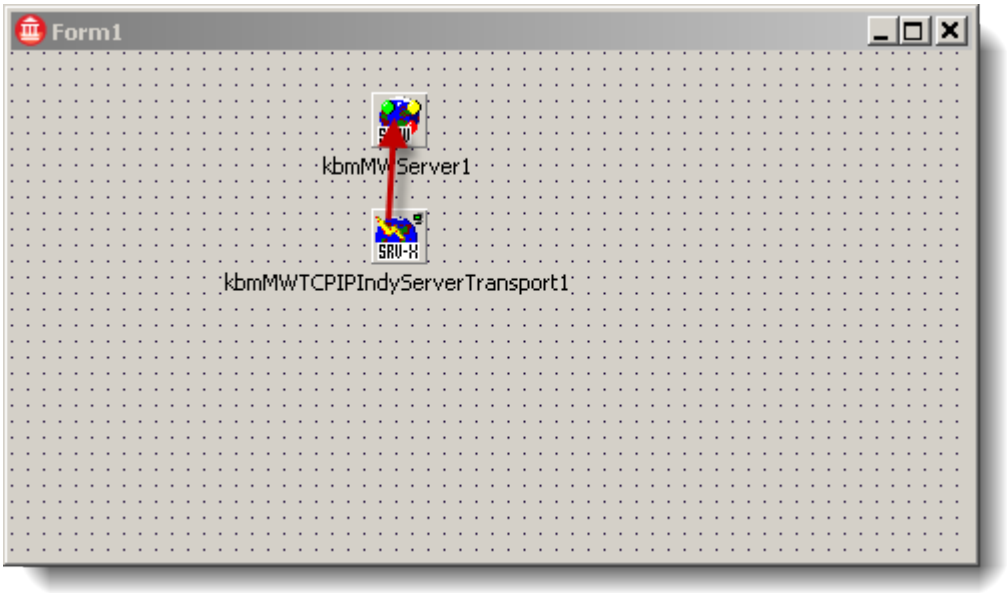
Creating a Webserver in 5 minutes using kbmMW!

kbmMW is a complete framework for building middleware/n-tier systems typically consisting of application servers and clients.

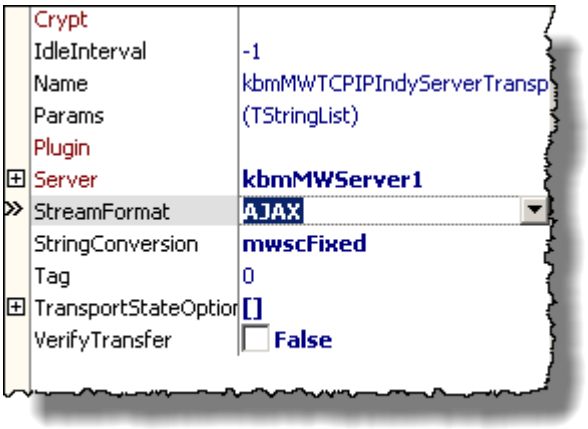
One of the many advantages of kbmMW is that it was designed, from ground up, to have a very flexbile, plugable and extendable architecture. That flexibility makes it possible to build a very capable kbmMW based web server in only 5 minutes, using kbmMW Professional or Enterprise Edition.

First we create a new standard VCL forms application. For real life scenarios, one would probably create a Windows service application instead. This sample can easily be converted to a service application instead, but its out of scope for this article to discuss that.

We will add a kbmMW application server and a kbmMW TCP/IP server transport and hook them up by setting kbmMWTCPIPIndyServerTransport1.Server to point at kbmMWServer1.

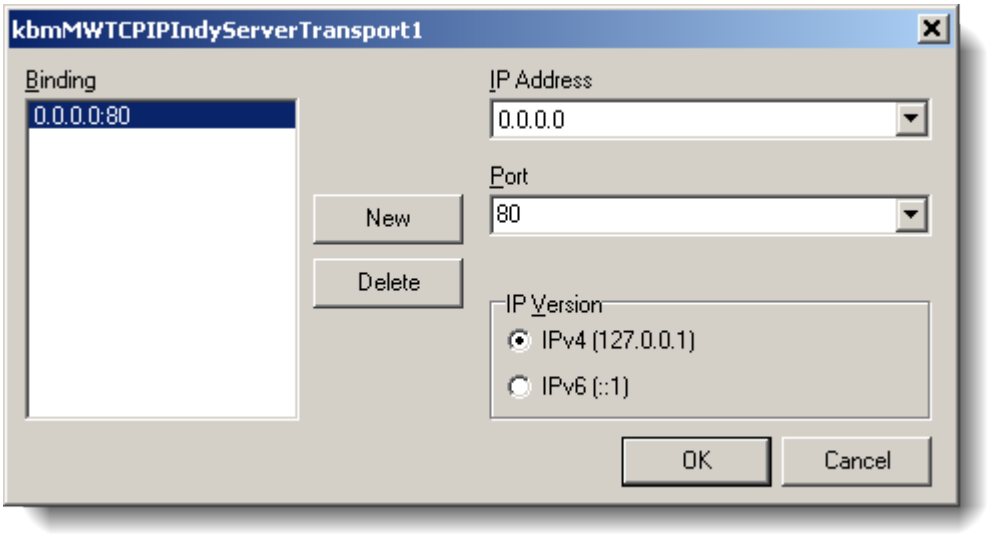


Next we set the transport components property Streamformat to AJAX.



Doing that ensures that the transport will understand clients sending even advanced HTTP requests to it.

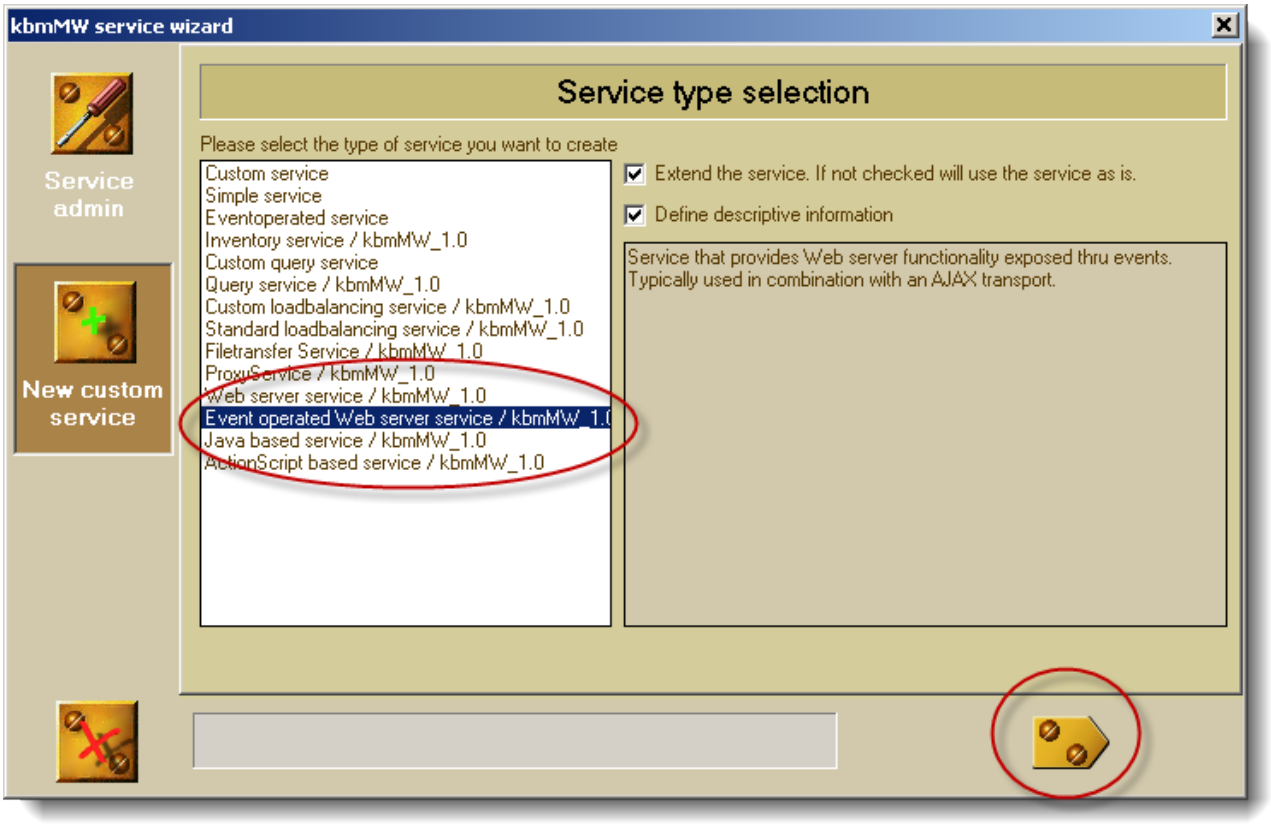
A standard web server will listen on port 80 for browsers connecting to it. Thus we configure our webserver to listen on port 80. It can be made to listen on multiple ports, if one would like it to also listen to 8080, 8000 etc too. We do that by clicking the Bindings property of the transport component, and then click New and set Port to 80.



Next step is to create a new kbmMW service that will handle requests from client browsers and produce web pages back to them. We use the wizard to create one (File/New/Other/Components4Developers/kbmMWService wizard).

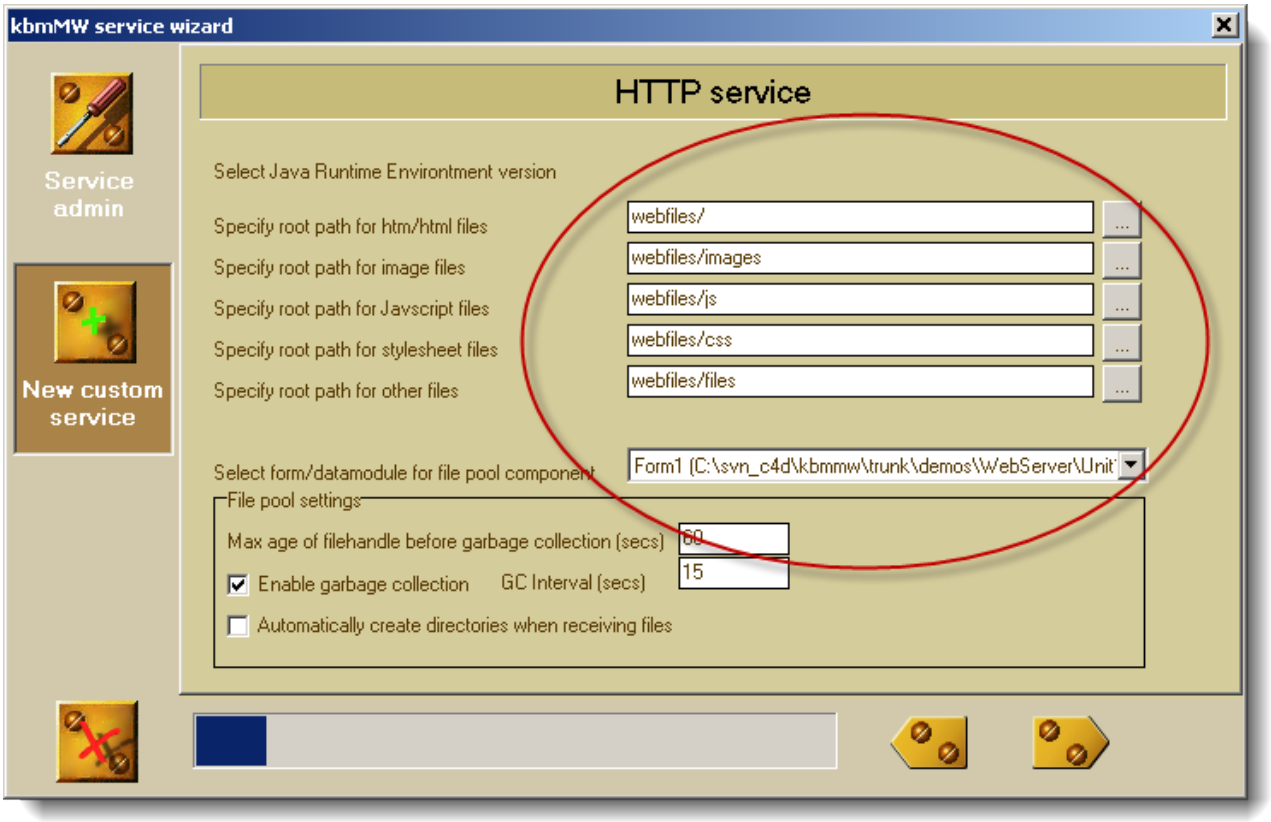
Lets select to create a service based on “Event operated web server service”.

It gives us some easy to use events that we optionally can take advantage of, however the service that is created when we are done with the wizard, is from outset able to service files from the disk directly to the browsers. Thus its already able to serve static HTML pages and files.



Click the Next button.

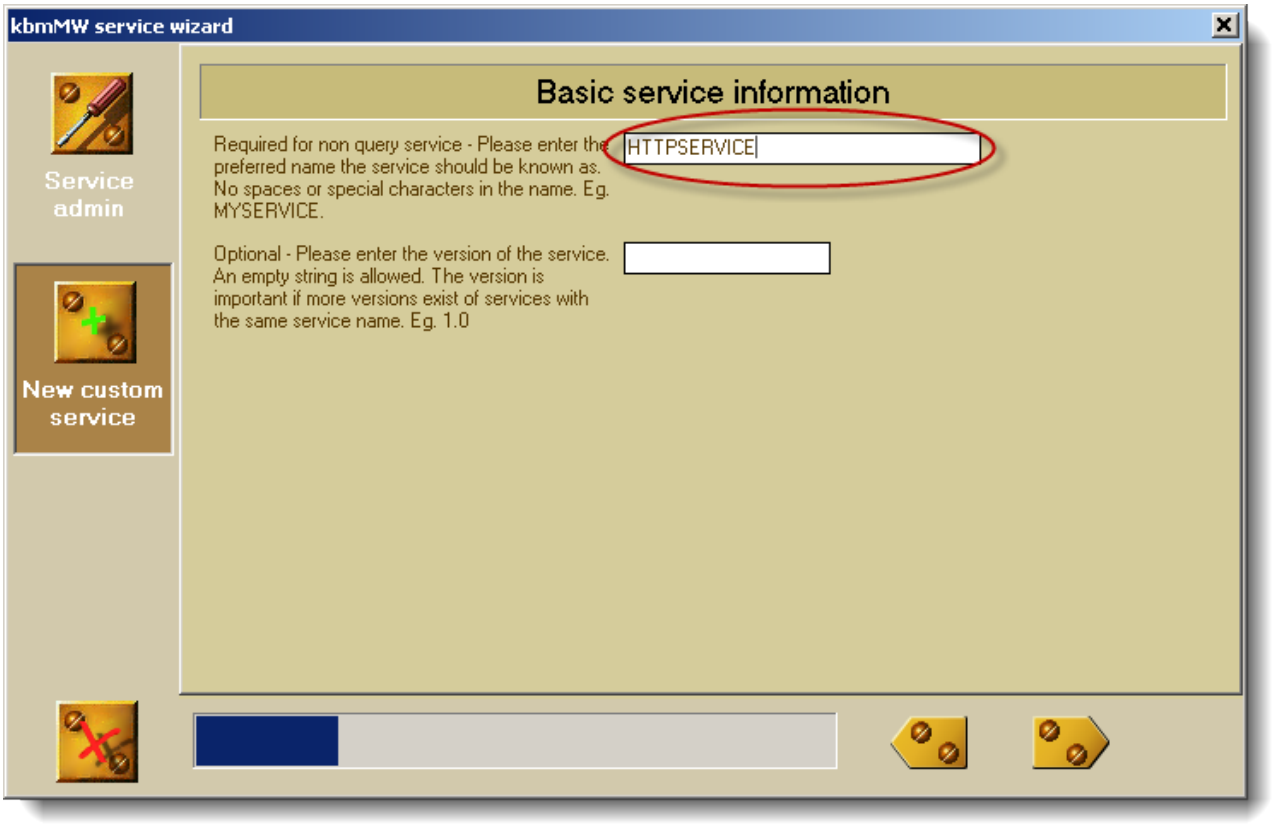
Now a configuration screen for the web kbmMW service is shown.



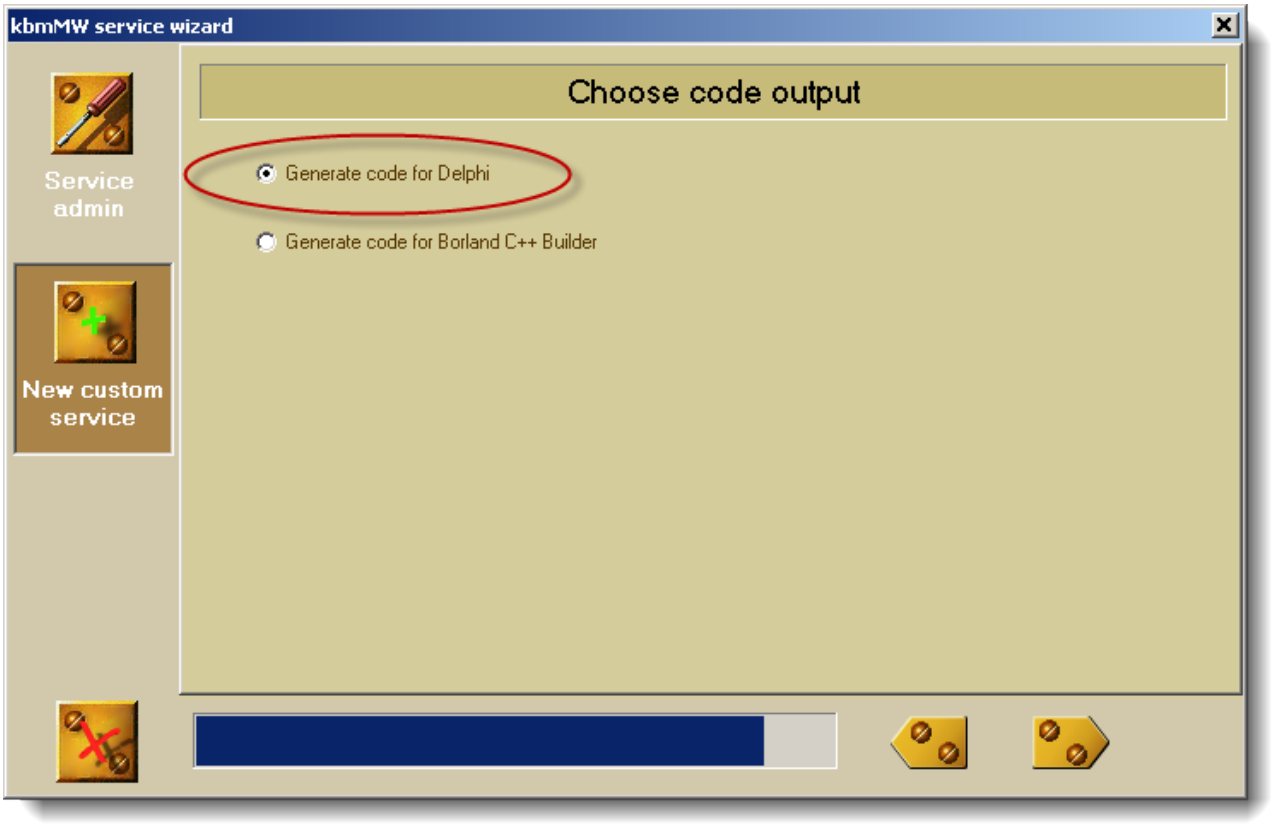
We specify here several things. First we tell the service where files are to be found, when a client request f.ex. html files, images, javascript, stylesheets and other files. 2ndly we specify that we would like the wizard automatically to place the relevant components in our project, to have the functionality needed to serve static files immediately. Thus we tell it that we want it to put a file pool on our main form (Form1). It's a form that is shared between all services and service instances. Only one Form1 will exist at any time in the lifetime of our web servers life.

Click next.

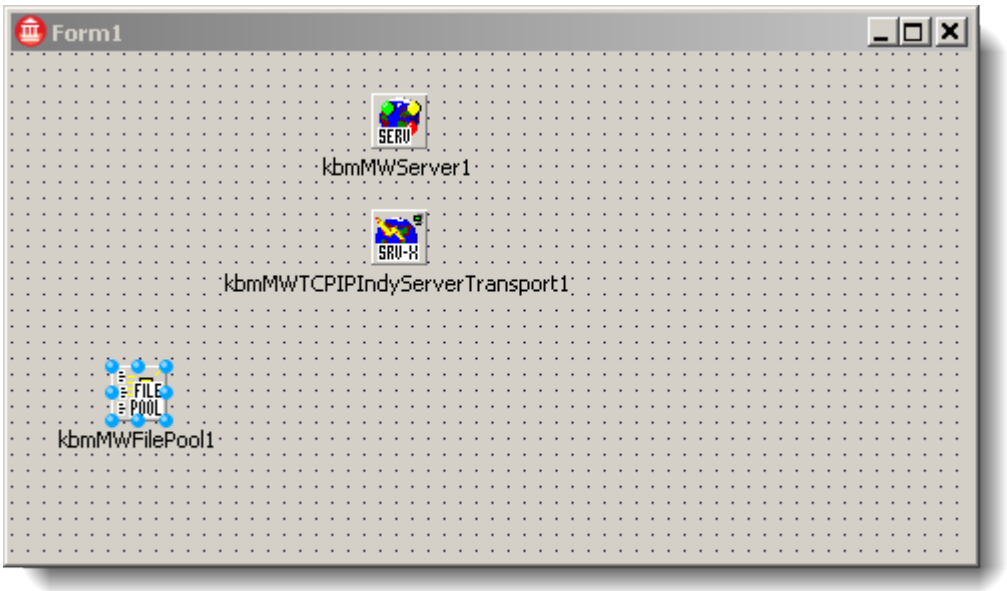
Now we must name our service. By convention we will give it the name HTTPSERVICE (no spaces or punctuation). That name is internally known by kbmMWs AJAX transport, so kbmMW will automatically refer all browser requests to this specific service.



Now click next all the way thru the next screens. They are optional and we don't need to fill out information on them. On last screen, select to create code for Delphi or C++Builder. In our sample we use Delphi, thus I have checked Delphi.

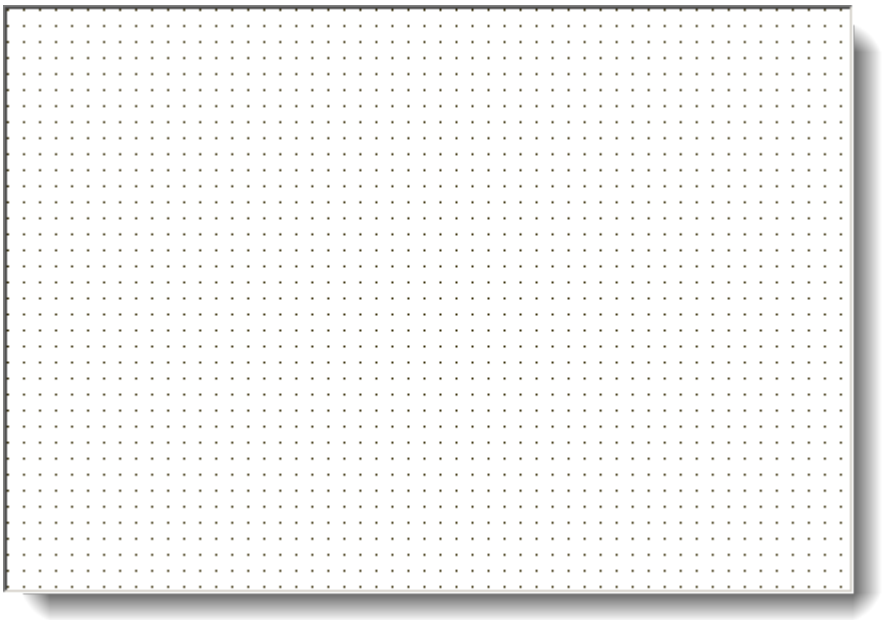


Click next and then the green checkmark button. Then the wizard updates our project with a new datamodule that will act as our web request handler, and a new component on our main form:

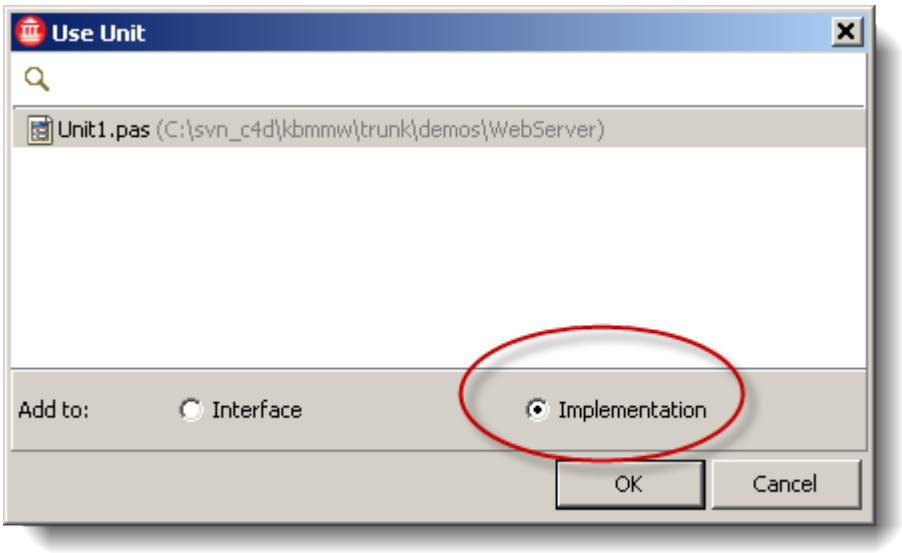


The kbmMWFilePool1 component will handle all file I/O access and ensure that noone is trying to update a file while another is reading it or visa versa. It also have file handle caching abilities which speeds up access to frequently used files.

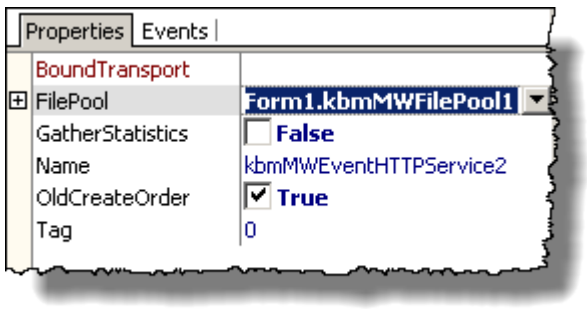
The new datamodule looks empty, but it already have a lot of functionality built in.



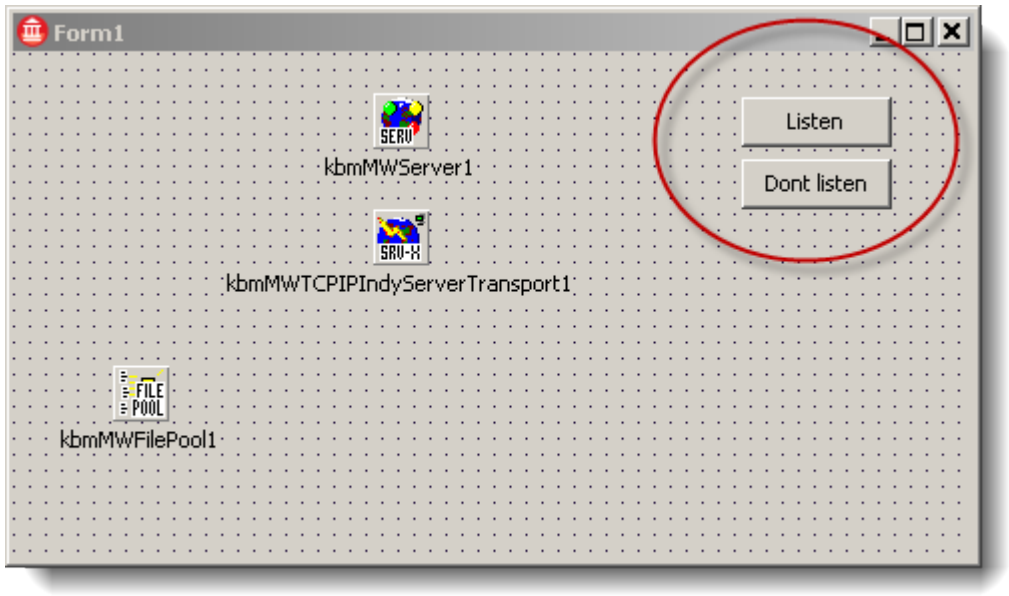
We need to set a property on the datamodule, but before doing that, we need to teach it to know about the main form. Click File/Use Unit, select Unit1 and click the Implementation radio button.



Then click OK. Next set the FilePool property of the datamodule to point at the filepool on the main form (Form1).



Now we put a couple of Tbutton's on the main form (Form1) to enable and disable the web server. Lets call the buttons btnListen and btnDontListen.



Double click btnListen and write this line of code:

```
kbmMWServer1.Active:=true;
```

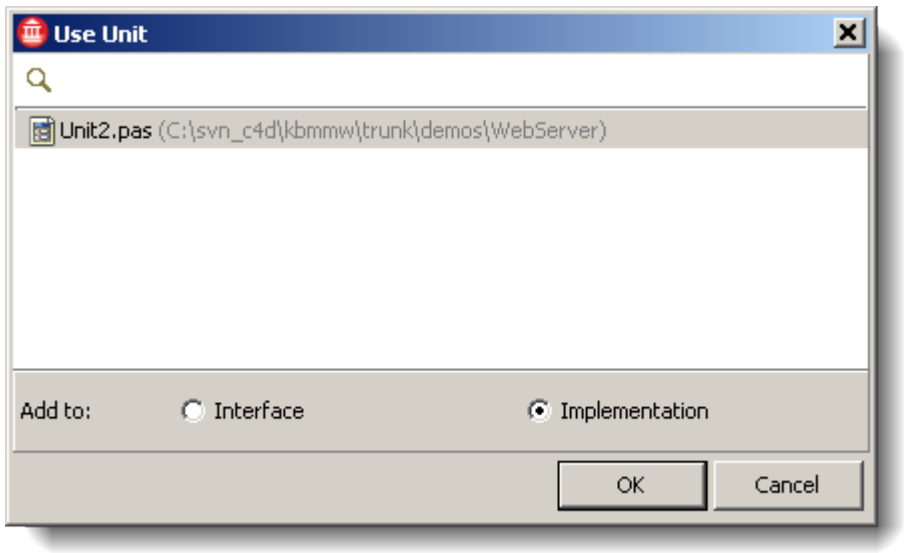
And double click btnDontListen and write this line of code:

```
kbmMWServer1.Active:=false;
```

Now we need to make the main unit Form1.pas aware about 2 other units that's needed. Locate the uses clause in the interface section, and add kbmMWAJAXTransStream and kbmMWCustomHTTPService to that clause:

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, kbmMWServer, kbmMWCustomTransport, kbmMWTCPIPIndyServerTransport,
  kbmMWFilePool, StdCtrls, kbmMWAJAXTransStream, kbmMWCustomHTTPService;
```

Finally we need to register the web service for kbmMWServer1. Click on File/Use Units, select unit2.pas, click the Implementation radio button and click ok.



Then doubleclick the main form (Form1) to create a form OnCreate event handler and write the following code:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  sd:TkbmMWHTTPServiceDefinition;
begin
    sd:=TkbmMWHTTPServiceDefinition(kbmMWServer1.RegisterService(TkbmMWEventHTTPService2,false));
    sd.RootPath[mwhfcHTML]:='webfiles/';
    sd.RootPath[mwhfcImage]:='webfiles/images';
    sd.RootPath[mwhfcJavascript]:='webfiles/js';
    sd.RootPath[mwhfcStyleSheet]:='webfiles/css';
    sd.RootPath[mwhfcOther]:='webfiles/files';begin
    kbmMWServer1.RegisterService(TkbmMWEventHTTPService2,false);
end;
```

In your setup, it may be that your service was named differently than in this sample (TkbmMWEventHTTPService2). In such case, just use the correct name.

Then compile the project. We now have a full featured web server having only written few lines of code.

Remember that the server expects to find html files in the subdirectory named webfiles. Crate that directory, and put an index.html file into it. For example something as simple as:

The server works!

Then start your web server exe, and click the Listen button.

When you then use your webbrowser to access 127.0.0.1 or localhost, you should see this result:

The server works!

The webserver is capable of acting as a AJAX server, and understands XML and can be brought to understand JSON too. For more information about that, read the whitepaper “kbmMW and AJAX” which can be found here:

http://www.components4programmers.com/downloads/kbmmw/documentation/kbmMW_and_AJAX.pdf