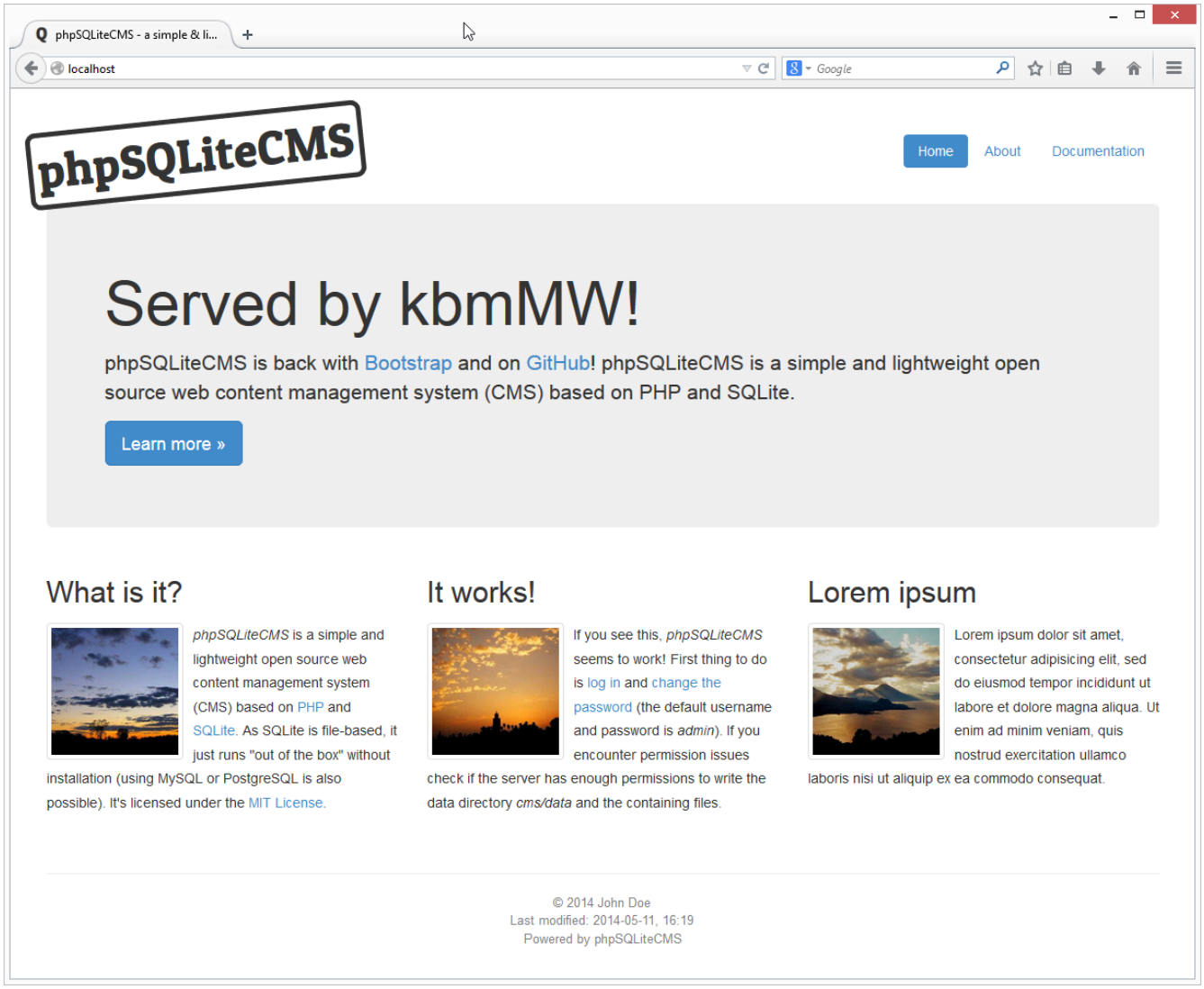
Serving an off the shelf Content Management System with kbmMW Professional or Enterprise Edition in less than 30 lines of code!

As other articles have described, kbmMW is a quite capable framework for building advanced, secure, high performance web applications with AJAX capability, containing all the features that are required of the most demanding web sites of today.

However the truth is also, that sometimes it's simply easier to use an open source CMS system, instead of building your own from scratch. Most such systems require a WAMP/LAMP setup installed. WAMP is an acronym for Windows/Apache/MySQL/PHP or Python, and LAMP is an acronym for Linux/Apache/MySQL/PHP or Python.

Installing and configuring Apache and MySQL in a safe, but usable way, require some tinkering with relatively lengthy configuration files.



Further you may want to take advantage of some features you already have in your existing infrastructure, which you may be running using kbmMW application servers.

Over the years, kbmMW has evolved to be the Swiss knife of n-tier solutions for Delphi/C++Builder/FPC, and since it has proven to be both stable, fast and easy to maintain, countless times in mission critical setups, where also life and death is involved, whats more natural than to use kbmMW in combination with existing open source software in a WMWP (Windows kbmMW PHP/Phyton) setup?

The recently released kbmMW Professional Edition and Enterprise Edition v. 4.50 includes many new features and improvements, of which I'll list a few important ones:

- 20 fold faster XML DOM parser. Probably one of the fastest, supporting namespaces, in the industry today.

- XSD (XML Schema Definition) parser and kbmMW object generator. It's able to parse even very complex XSD schemas and produce all the relevant object classes and types, that can be used to parse and generate XML AND JSON files in one single line of code.
- FastCGI support via kbmMW's web server services.

We will in this article primarily look at the FastCGI support, as that is what makes it possible to host a PHP open source CMS application like phpSQLiteCMS, in your kbmMW application server.

The kbmMW application server

If you have read other articles about the kbmMW application server, you will know that it's designed from ground up to be extremely flexible, allowing to put together building blocks to achieve the desired functionality.

The building blocks within kbmMW, we will be working with in this article, are the transport, transport stream format, service and the FastCGI extension.

The transport is basically a component that defines how external clients should connect to the kbmMW application server. In our case we want to make a kbmMW application server that acts like a web server, so we will need to use a transport that supports TCP/IP.

The transport stream format, defines how bytes being transmitted and received via the transport, should be interpreted. Again, we want a web server functionality, so in this case we should choose kbmMW's AJAX transport stream format.

The service is a kbmMW definition for a specialized Delphi/C++ datamodule, which contains relevant business logic. kbmMW comes with a good number of predefined services, that can be extended by the developer. One of them is a web service (not to confuse with SOAP which kbmMW ofcourse also supports, but that's another story). The web kbmMW service, understands GET/POST and other HTML functions, along with HTTP cookies, headers, authorizations etc. and thus have all the functionality to make a kbmMW based web server, with the correct transport and transport stream format.

The FastCGI extension is a kbmMW feature that allows kbmMW to communicate and even spawn FastCGI servers. A FastCGI server is an external process, which can either live on the same machine as the kbmMW application server, or on entirely different machines. Read more about FastCGI here: http://www.fastcgi.com/drupal/

The point of a FastCGI server is basically to be ready to receive data from a web server, process the data and return HTML to be sent to a browser. In earlier days, CGI (Common Gateway Interface) was the norm to use. It was an executable that was spawned by the web server, processed data sent to it via its stdin and environment variables, and sent data out via its stdout and stderr, which were picked up by the web server and sent as

result to the browser. After each call to the CGI executable, the executable died and needed to be respawned next time a request would require data from it. That was not very performance friendly, why FastCGI was invented. A FastCGI server continues to run, after being spawned, and all communication with it is typically done via TCP/IP. A FastCGI server can typically process requests from multiple web server calls at the same time. kbmMW do not support CGI applications.

In our sample we will use a PHP FastCGI server.

As multiple transports with different transport stream formats, can be added to any kbmMW application server, it's very easy to extend your existing kbmMW application server with for example web functionality.

Hence no need to add new complexity to your company's infrastructure. You can simply extend your existing kbmMW application server with the new requirements.

Prerequisites

phpSQLiteCMS is required for our sample. It can be downloaded from here http://phpsqlitecms.net/

After you have downloaded it, extract all its contents to a subdirectory named *phpsqlitecms* somewhere on your computer. Remember the directory as you will need to refer to it later.

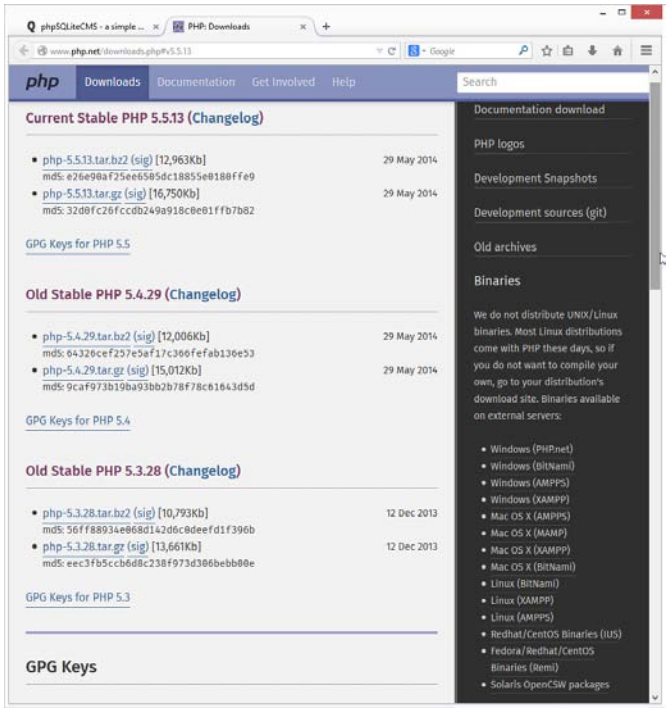
You could choose any other PHP, Python, Ruby or Perl (which all provides FastCGI servers) based CMS system for your setup, but be prepared to read about how to configure it. If you choose a PHP based CMS, this sample will generally be ready to use it with very little change.

Here is a selection of CMS systems.

http://en.wikipedia.org/wiki/List_of_content_management_systems and here is a site comparing CMS systems: http://www.opensourcecms.com/

As the sample is using a PHP based CMS system, we obviously also need to download and install PHP from http://www.php.net/

If you click on the Download link, you will find a section of links to the right of the page, where you can download Windows binary (precompiled) installers.



After downloading the Windows installer, run it.

You will typically end up with a PHP directory somewhere (in my case in C:\PHP).

An important next step is to configure PHP. Which exact configuration is needed will depend on the various PHP applications that you may choose to run. As we, in this sample, run phpSQLiteCMS, I'll explain what to configure for that particular CMS.

Configuration of PHP is done by creating a valid php.ini file in the PHP installation directory (the directory also contains php.exe and php-cgi.exe). Easiest way to define a php.ini file is to copy either php.ini-development or php.ini-production and make modifications to it. After coping one of the files and naming it php.ini, open php.ini in notepad.

Now search for ***extension_dir = "ext"***

And remove the ; to uncomment the line.

Then search for ***extension=php_mbstring.dll*** and ***extension=php_pdo_sqlite.dll***

And remove the ; to uncomment the lines.

Then search for ***session.save_path = "/tmp"***

And remove the ; to uncomment the line. Further change the line to

session.save_path = "C:\PHP\sessionfolder"

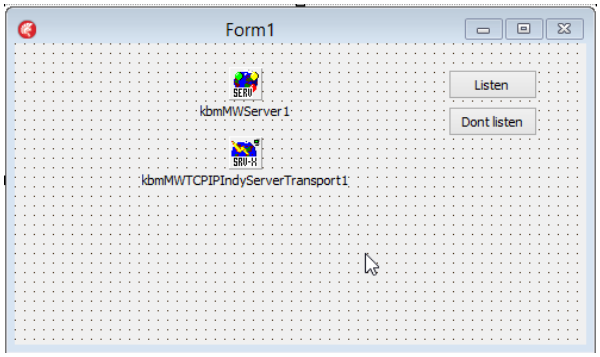
or whatever directory is matching your PHP installation. The sessionfolder directory may not exist, so you should manually create it.

That's all that is needed to configure on PHP.

A kbmMW web server

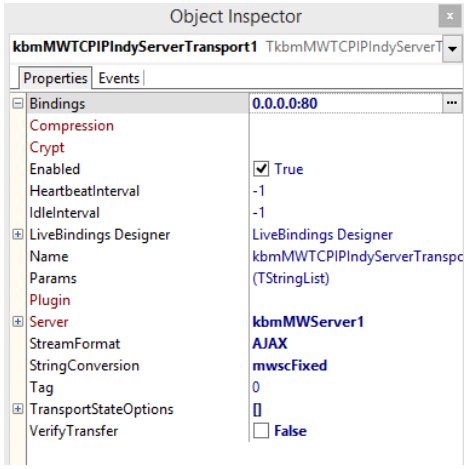
So let's get our hands dirty and produce a kbmMW application server that acts as a web server. For the sake of simplicity, the application server will be created as a regular VCL application. However it's just as easy to create it as a Windows Service, which can run the moment the Windows machine starts up.

A kbmMW application server must at least have a central module or form (in this sample), which hosts a TkbmMWServer instance and one or more kbmMW server side transport components. Other central components may also be placed here as needed, like database connection pools, authorization manager components etc. but in our sample we only need a TkbmMWServer and a TkbmMWTCPIPIndyServerTransport component at first, perhaps along with a couple of buttons to start/stop the application server.



We will leave all properties default for the kbmMWServer1 component, but will need to configure a couple of properties on the transport component.

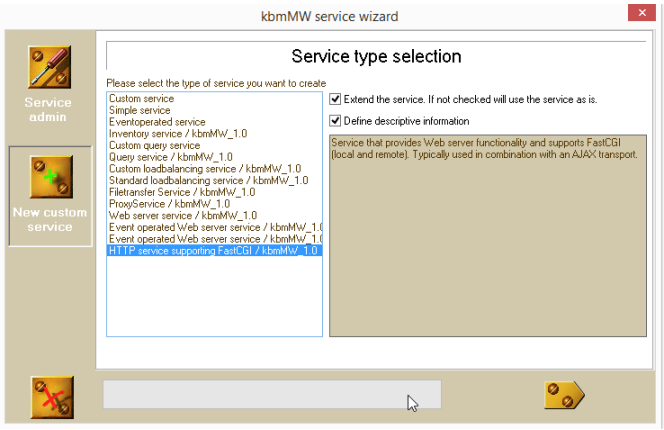
First add a binding via the Bindings property. Set its IP address to 0.0.0.0 and the port to 80. It means that the server transport will listen for client connections coming in via all network cards on port 80, which is the port used by web browsers by default.



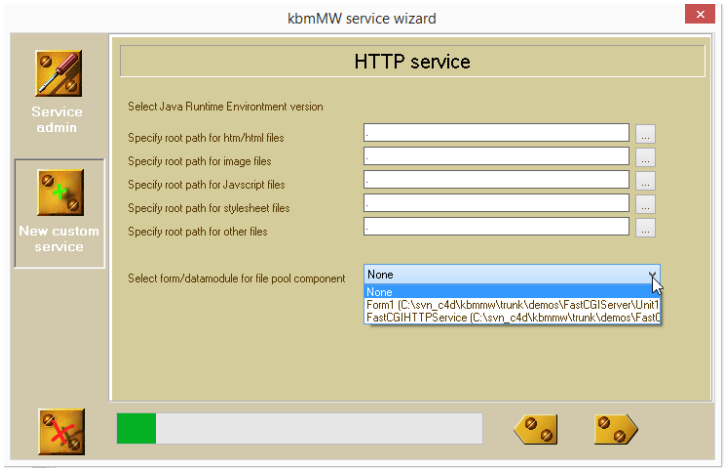
Then set the StreamFormat to AJAX and VerifyTransfer to false.

Next we need to add a new web kbmMW service. It's very easy to do, by using the kbmMW Service Wizard, which you will find in File/New/Other, select Components4Developers and the kbmMW Service Wizard.

When it opens, it shows a list of various known service types you can choose between. A couple of them are for creating web functionality, but the one we are going to use is the "HTTP service supporting FastCGI…".



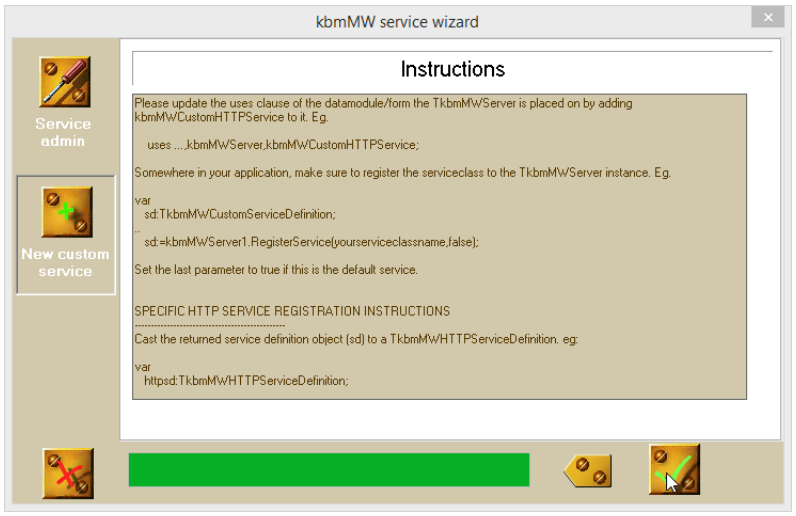
Then click the "Next step" button at the bottom right corner.



Leave the first “path” edits as is. We will deal with them later. However we can save a little bit of time, by selecting which form/datamodule, should contain a file pool component. The file pool component is a component that structures access to regular files in the file system, and also cache access to them to speed up things. As a traditional web server typically serves many files, like html, css, php, js etc files, from disk, the web server needs a file pool.

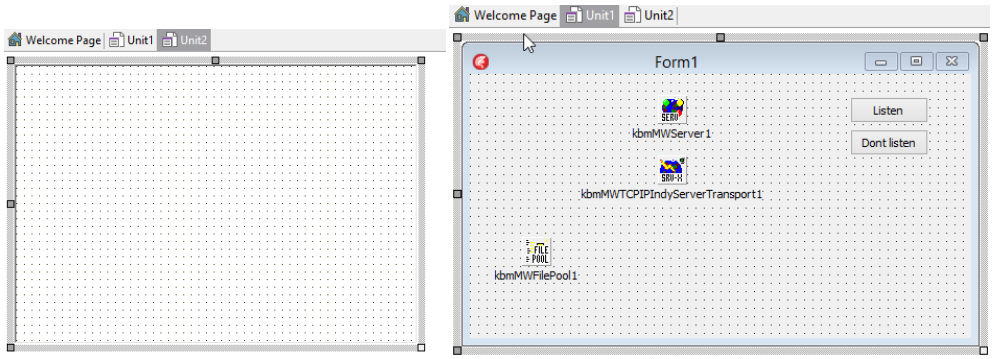
Select Form1, and leave the new options default.

Then click “Next step” a number of times without changing anything until you reach this page:

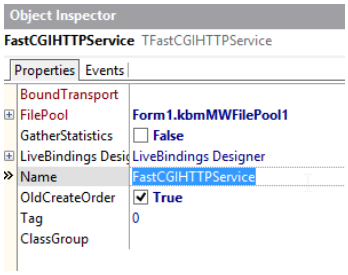


And finally click the generate button with the green checkmark.

Now a new specialized, although seemingly empty, datamodule will be generated for you, and the wizard will have added a TkbmMWFilePool component to your main form.



Add Unit1 to Unit2's implementation section uses clause, and then set the property FilePool to of Unit2's datamodule to point to kbmMWFilePool1 on the main unit (Unit1).



Now the bulk of those 30 lines of code will need to be written. Double click the main form to create its OnCreate event.

Add Unit2 to Unit1's implementation section uses clause.

Then write the following code in the FormCreate event handler (you can omit comments ☺).

```
procedure TForm1.FormCreate(Sender: TObject);
var
   sd:TkbmMWHTTPFastCGIServiceDefinition;
   sl:TkbmMWConnectionStringArray;
begin
     sd:=TkbmMWHTTPFastCGIServiceDefinition(kbmMWServer1.RegisterService(TFastCGIHTTPService,false));

     // Its possible to define a default file name if no file is referenced.
     // As we define a PHP FastCGI site, index.pbp is typically the default.
     // Its also possible to define the default via an URL rewrite. See further down.
     //     sd.DefaultFile:='index.php';

     // Define the root of all resources for the PHP site.
     // Actual .php files are placed in the mwhfcOther category.
     // Typically these values all points to the same PHP webapp root directory.
     // The directory is seen from the current directory of the kbmMW server.
     sd.RootPath[mwhfcHTML]:='phpsqlitecms';
     sd.RootPath[mwhfcImage]:=sd.RootPath[mwhfcHTML];
     sd.RootPath[mwhfcJavascript]:=sd.RootPath[mwhfcHTML];
     sd.RootPath[mwhfcStyleSheet]:=sd.RootPath[mwhfcHTML];
     sd.RootPath[mwhfcOther]:=sd.RootPath[mwhfcHTML];

     // Most FastCGI servers require these settings to be made.
     // ServerName and ServerPort is the name/port for which the server
```

```
    // is accessed from the "outside" before any NAT translations or proxying.
    sd.ServerName:='192.168.1.103';
    sd.ServerPort:=80;

    // Define how to connect to FastCGI servers serving this PHP application.
    // Here we define one local PHP FastCGI server (the group name is not
    // relevant as such but is used to separate different FastCGI instances
    // from eachother).
    // More can be defined as separate groups. kbmMW will then round robin
    // loadbalance requests to them.
    // As a minimum IP and Port entries must be defined for a FastCGI server.
    // Optionally Launch and Parameters can be defined, to let kbmMW automatically
    // start FastCGI servers as needed.
    sl:=TkbmMWConnectionStringArray.Create;
    try
       sl.Add('<|local|>');
       sl.Add('IP=192.168.1.103');
       sl.Add('Port=8764');
       sl.Add('Launch=c:\PHP\php-cgi.exe');
       sl.Add('Parameters=-b 192.168.1.103:8764');

       // Rewrite any url for files not found.
       sd.Rewrites.Add('^/cms/(.*)$','/cms/index.php?qs=$1',[mwhuroIfFileNotExists,mwhuroBreak]);
       sd.Rewrites.Add('^/(.*)$','/index.php?qs=$1',[mwhuroIfFileNotExists,mwhuroBreak]);

       sd.FastCGIModules.RegisterFastCGIModule(KBMMW_FASTCGI_URL_MASK_PHP,sl,
          'C:\svn_c4d\kbmmw\trunk\demos\FastCGIServer\phpsqlitecms'
          );
    finally
       sl.Free;
    end;
end;
```

Whats the code doing?

```
    sd:=TkbmMWHTTPFastCGIServiceDefinition(kbmMWServer1.RegisterService(TFastCGIHTTPService,false));
```

is registering the newly created web kbmMW service to the TkbmMWServer instance. You can, and typically will, have many kbmMW service modules registered. Some for database access, some for business code and some for web code. In our example we only need one service.

The registerservice call returns a service definition instance. It can optionally be used to configure service related settings. Like how many instances of the service is allowed to run at any time, and in this case, also information about where to find web related files like html, js, images etc.

In this sample, we assume there will be a subdirectory named *phpsqlitecms* under the directory from which the kbmMW application server executable is run. That subdirectory will be the one containing the open source CMS. If you have installed the CMS in another directory, please write the full path to that directory instead.

```
    sd.ServerName:='192.168.1.103';
    sd.ServerPort:=80;
```

These lines are required for most FastCGI based sites. The CMS PHP code will use these values when it generates dynamic links in the HTML that it produces. The ServerName/ServerPort is the DNS name (or IP address) and TCP/IP port that should be used when accessing the kbmMW web server from a browser. If you have NAT translation configured, it should be the external IP address or DNS name that should appear here,

and typically port 80 (or 443 if you have defined that the kbmMW application server should support SSL. This is not described in this article).

```
sl:=TkbmMWConnectionStringArray.Create;
try
   sl.Add('<|local|>');
   sl.Add('IP=192.168.1.103');
   sl.Add('Port=8764');
   sl.Add('Launch=c:\PHP\php-cgi.exe');
   sl.Add('Parameters=-b 192.168.1.103:8764');
```

This part defines how to access (and automatically spawn) the PHP FastCGI server on the local machine. The IP address and port should be the internal IP address used for the kbmMW application server to contact the FastCGI php-cgi.exe executable. The parameters property defines parameters for the php-cgi.exe file, and basically tells it what IP address and port it should serve. If the php-cgi FastCGI server runs on another machine, you should only define the IP= and Port= lines to point to the relevant FastCGI server.

If you have multiple FastCGI servers running, which all are able to serve the PHP CMS site you can define them all separated by a line containing a unique name in the form '<|uniquename|>'. In our case we are using a CMS that use SQLite for datastorage. And as SQLite is a flatfile local database, each server have its own separate SQLite database, why it doesn't make sense to define multiple FastCGI servers for our sample.

```
   // Rewrite any url for files not found.
   sd.Rewrites.Add('^/cms/(.*)$','/cms/index.php?qs=$1',[mwhuroIfFileNotExists,mwhuroBreak]);
   sd.Rewrites.Add('^/(.*)$','/index.php?qs=$1',[mwhuroIfFileNotExists,mwhuroBreak]);
```

kbmMW v. 4.50 now also supports URL rewrites, which is almost a requirement for most Apache based web sites. The purpose of an URL rewrite is to convert specific URLs (defined with a regular expression pattern) to some other URL that makes more sense for the web server and FastCGI servers. Typically the rewrites are needed in the first place to reduce more complex URLs to simpler ones, that are more SEO friendly.

In this case kbmMW will rewrite any URLs starting with /cms/something to /cms/index.php?qs=something and any remaining URLs to /index.php?qs=whatever. The order of the rewrite definitions are important, as they are checked in order. The ***mwhuroIfFileNotExists*** option ensures that if the given URL actually points to a true existing file, it will not be rewritten, otherwise it will. The ***mwhuroBreak*** option specifies that if a pattern match is made, then no more pattern matches will be done for this particular URL.

```
   sd.FastCGIModules.RegisterFastCGIModule(KBMMW_FASTCGI_URL_MASK_PHP,sl,
      'C:\svn_c4d\kbmmw\trunk\demos\FastCGIServer\phpsqlitecms'
      );
```

Finally we register the FastCGI module configurations. The constant KBMMW_FASTCGI_URL_MASK_PHP is a regular expression mask that match a file that is to be considered eligible for being run by this particular FastCGI server. Other constants for Ruby, Perl and Python are also available. The final argument is a local root path for the locally spawned FastCGI server. Change to match your setup.

```
finally
   sl.Free;
end;
```

Now we just need to add a line or two of codes to the listen and don't listen buttons and also stop the server the moment the application closes, then we are done.

Put event handlers for the btnListen.OnClick and btnDontListen.OnClick events, and for the Form1.OnClose event:

```
procedure TForm1.btnDontListenClick(Sender: TObject);
begin
     kbmMWServer1.Active:=false;
end;

procedure TForm1.btnListenClick(Sender: TObject);
begin
     kbmMWServer1.Active:=true;
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
     kbmMWServer1.Active:=false;
end;
```

Compile and run the kbmMW application server, and you have a new cool CMS system readily running.

This sample can of course be extended in many ways, like intercepting some of the data access calls, that instead could lookup data in your corporate databases and serve those as JSON or dynamically built HTML data.

For more information about that, please refer to older articles in Blaise Pascal magazine.

Whats also cool, is that the FastCGI extension is designed to work on ALL platforms, which means you could create an IOS kbmMW application server which supports FastCGI. The actual FastCGI server must of course already be running on another computer. Only kbmMW application servers with FastCGI on Windows can automatically launch FastCGI servers.

Happy FastCGI'ing!

Best regards

Kim Madsen

Components4Developers